



**SUITABILITY OF UNIDATA METAPPS FOR
INCORPORATION IN PLATFORM-INDEPENDENT
USER-CUSTOMIZED AVIATION WEATHER
PRODUCTS GENERATION SOFTWARE**

THESIS

Harmen P. Visser, Captain, USAF

AFIT/GM/ENP/02M-09

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Report Documentation Page

Report Date 8 Mar 02	Report Type Final	Dates Covered (from... to) Jun 2001 - Mar 2002
Title and Subtitle Suitability of Unidata Metapps for Incorporation in Platform-Independent User-Customized Aviation Weather Products Generation Software	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Capt Harmen P. Visser, USAF	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB OH 45433-7765	Performing Organization Report Number AFIT/GM/ENP/02M-09	
Sponsoring/Monitoring Agency Name(s) and Address(es) AFWA/DNXT ATTN: Mr. Bruce Telfeyan 106 Peacekeeper Drive Offutt AFB NE 68113-4039	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes		

Abstract

Due to multiple factors, including an increase in military operations tempo and the improved resolution of meteorological models, demand for access to customized aviation weather products has increased exponentially. This has given rise to a need for a multi-purpose interactive aviation weather product generation software solution. This software solution must be platform-independent, multiple data source access configurable, robust, extensible or upgradeable, user-friendly, and an improvement over current visualization applications used in the operational military aviation weather community. This thesis determines whether Unidata MetApps meets these criteria. A software reuse and component-based engineering approach was taken in this thesis. Two experimental applications were constructed using a software design approach resembling the Facade software design pattern. The first application used existing MetApps stand-alone prototype applications, while the second exploited capabilities of the MetApps component library. Both experimental applications were measured against the above set of criteria to determine their suitability for incorporation in platform-independent user-customized aviation weather products generation software. The results prove that a Facade software design approach can be effectively used to build applications. It was determined however that, even though MetApps shows promise, it may not be suitable for incorporation into an operational application.

Subject Terms

Meteorology, Meteorological Applications (MetApps), Interactive Graphics, Java, Unidata.

Report Classification

unclassified

Classification of this page

unclassified

Classification of Abstract

unclassified

Limitation of Abstract

UU

Number of Pages

103

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government.

AFIT/GM/ENP/02M-09

SUITABILITY OF UNIDATA METAPPS FOR INCORPORATION IN PLATFORM-
INDEPENDENT USER-CUSTOMIZED AVIATION WEATHER PRODUCTS
GENERATION SOFTWARE

THESIS

Presented to the Faculty
Department of Engineering Physics
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Meteorology

Harmen P. Visser, BS

Captain, USAF

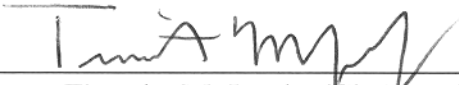
March 2002

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

SUITABILITY OF UNIDATA METAPPS FOR INCORPORATION IN PLATFORM-
INDEPENDENT USER-CUSTOMIZED AVIATION WEATHER PRODUCTS
GENERATION SOFTWARE


Harmen P. Visser, BS
Captain, USAF

Approved:



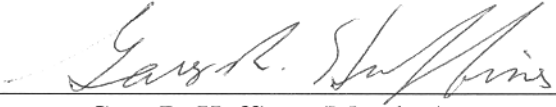
Timothy M. Jacobs (Chairman)

28 FEB 02
date



Michael K. Walters (Member)

28 Feb 02
date



Gary R. Huffines (Member)

28 Feb 2002
date

Acknowledgements

I would like to express my sincere appreciation to my thesis advisor, Lt Col Timothy Jacobs, for his guidance and support throughout the course of this thesis effort. The insight and experience he provided was greatly appreciated. My gratitude also goes out to Lt Col Walters, for his dry wit, and Major Huffines, for helping me keep things that are truly important in perspective. I would, also, like to thank my organizational sponsor, the Technology Exploitation Branch at the Air Force Weather Agency, for both the technical support and latitude provided to me in this endeavor. I also wish to acknowledge the help from my fellow thesis students from both the Engineering Physics and Electrical and Computer Engineering Departments. Their comments, suggestions, quips, snipes, good-natured haranguing, and occasional much needed “correction-of-attitude” helped keep me from always losing my temper.

I am, also, indebted to the late author Theodore S. Geisel. At the age of ten, I was presented by my parents with a copy of Geisel’s “The Sneetches and Other Stories,” and introduced to the English language (some say that this goes a long way to explaining why I write and speak English the way that I do).

Finally, and certainly not least, I wish to thank the Lord and my family. Without their patience, understanding, and unconditional love, I would certainly have had a lot tougher time getting through the creation of this thesis than I did.

Harmen P. Visser

Table of Contents

	Page
Acknowledgements.....	iv
List of Figures.....	vii
List of Tables	viii
Abstract.....	ix
1. Introduction.....	1
1.1 Background.....	1
1.2 Statement of the Problem.....	5
1.3 Scope.....	6
1.4 Overview of Approach.....	6
1.5 Organizational Overview	8
2. Related Work	10
2.1 Overview.....	10
2.2 McIDAS.....	10
2.3 GrADS	13
2.4 Vis5D.....	15
2.5 VisAD	17
2.6 MetApps.....	21
2.7 4DWX.....	23
2.8 Summation	25
3. Methodology	27
3.1 Overview.....	27

3.2 Goals	27
3.3 Requirements for Success	28
3.4 Design of Experimental Applications	29
3.4.1 Overview	29
3.4.2 Application Design Approach 1	31
3.4.3 Application Design Approach 2	34
3.5 Experimental Programming Environment Setup	37
3.6 Data Requirements and Handling	40
4. Results	44
4.1 Overview	44
4.2 Application Design Approach 1	44
4.3 Application Design Approach 2	49
4.4 Summary of Results	52
5. Conclusions and Recommendations	54
Appendix A: Abbreviations	58
Appendix B: Unidata MetApps Detailed Analysis	60
Appendix C: Application Source Code	72
Bibliography	89
Vita	92

List of Figures

Figure	Page
1. McIDAS Product	11
2. GrADS Meteorogram.....	14
3. Vis5D Screenshot.....	16
4. VisAD Screenshot.....	18
5. MetApps Screenshot	22
6. 4DWX VMET Screenshot.....	25
7. Standard, simplified view of the Facade software design pattern	31
8. Application 1 Diagram.....	33
9. Application 2 Diagram.....	36
10. Experimental Application Screenshot.....	45
11. Interactive Sounding Screenshot.....	63
12. GDV Screenshot	65
13. Image Viewer Screenshot	67
14. Image Viewer Class Diagram	68
15. MetApps Library Package Diagram.	70

List of Tables

Table	Page
1. Experimental Software Environment.....	38
2. Experimental Hardware Environment	40
3. Additional Libraries Required for Design Approach 2.....	35
4. Summary of Experimental Results	53

Abstract

Due to multiple factors, including an increase in military operations tempo and the improved resolution of meteorological models, demand for access to customized aviation weather products has increased exponentially. This has given rise to a need for a multi-purpose interactive aviation weather product generation software solution. This software solution must be platform-independent, configurable for multiple data source access, robust, extensible or upgradeable, user-friendly, and an improvement over current visualization applications used in the operational military aviation weather community. The purpose of this thesis is to determine whether Unidata MetApps meets these criteria.

A software reuse and component-based engineering approach was taken in this thesis. Two experimental applications were constructed using a software design approach resembling the Facade software design pattern. The first application used existing Unidata MetApps stand-alone prototype applications, while the second exploited capabilities of the MetApps component library. Both experimental applications were measured against the above set of criteria to determine their suitability for incorporation in platform-independent user-customized aviation weather products generation software. The results prove that a Facade software design approach can be effectively used to build meteorological applications. It was determined however that, even though MetApps shows promise, it may not be suitable for incorporation into an operational meteorological software solution.

SUITABILITY OF UNIDATA METAPPS FOR INCORPORATION IN PLATFORM- INDEPENDENT USER-CUSTOMIZED AVIATION WEATHER PRODUCTS GENERATION SOFTWARE

1. Introduction

1.1 Background

In 1995, the Air Force Weather Information Network (AFWIN) came online. Under the direction of the Air Force Global Weather Central Product Improvement Branch (the precursor of today's Air Force Weather Agency Technology Exploitation Branch) AFWIN provided access to Relocatable Window Model (RWM) products produced by automated Mesoscale Gridded Window Display System (MGWDS) sessions. The RWM was superseded in 1998 by the National Center for Atmospheric Research/Pennsylvania State University Mesoscale Model 5 (MM5), which is still running today. MM5 products were thenceforth produced directly for distribution to customers via AFWIN. The Air Force Weather Agency Technology Exploitation Branch produced displays for hundreds of standard and derived meteorological parameters for weather forecaster use worldwide. By 1999, as the MM5 began to include more theaters, the number of pre-staged automated weather products had expanded to over 350,000 per day (Telfeyan, 2001). This consisted of a huge archive of Graphics Interchange Format (GIF) images, which were completely recycled every 24 hours. A large percentage of the 350,000 images were never viewed by anyone. Producing 350,000 archived images per

day was an inefficient use of computing resources and it became imperative that the large number of pre-staged automated weather products be reduced.

The concept was introduced of providing interactive interfaces on AFWIN that would allow an Internet user to specify what they needed and have the Air Force Weather Agency computing resources produce the required charts upon demand. An interface was developed called Interactive Meteorogram and Skew T (IMaST). A meteorogram is a chart in which one or more meteorological variables are plotted against time while a skew T (an abbreviation of skew T - log p) is a standard plot used by meteorologists to analyze data from a sounding. IMaST allowed a user to produce a meteorogram or skew T diagram for any location within the geographical domains of various meteorological forecast models including the MM5. The user was prompted to enter either latitude and longitude coordinates, an ICAO (International Civil Aviation Organization) four-letter identifier, or point and click a location on a number of theater maps using a mouse.

Air Force weather went through reorganization in the late 1990's. As part of this reorganization, the production of many theater-specific aviation weather products was decentralized and transferred to the newly created Air Force regional weather hubs (Sembach, Shaw, etc.). However, responsibility for the generation of user-customized meteorological products using meteorological model data was left with the Air Force Weather Agency Technology Exploitation Branch. The reason for this was the requirement for large computing resources to generate the required meteorological forecast model data files and the configuration complexities involved in running the necessary Grid Analysis and Display System (GrADS) software with an associated JavaScript interface. Thus there remained the problem of a single point of failure. The

regional Air Force weather hubs have a wide variety of different computing hardware at their disposal, consisting mostly of Microsoft Windows based personal computers (PC's) and some UNIX workstations. Compared to the Air Force Weather Agency Technology Exploitation Branch, the weather hubs' computing resources are more limited.

In 2001, the Interactive Gridded Analysis and Display System (IGrADS) was developed to eventually replace IMAST. IGrADS expanded the capabilities of IMAST to include vertical cross-sections, alphanumeric model output, two-dimensional weather charts with user-selected parameters, severe weather meteorograms, etc. IGrADS replaced the IMAST JavaScript interface with one based on Java™ classes. This IGrADS feature takes advantage of the built-in capabilities of modern Java™-enabled Internet content browsers such as Netscape and Internet Explorer 5.0 to run Java™ applications over the Internet. Nonetheless, GrADS was still the behind-the-scenes engine that produced all the user-requested meteorological products distributed via IGrADS.

The Air Force Weather Agency Technology Exploitation Branch currently depends on IGrADS to provide user-customized meteorological products to its customers through its Joint Air Force and Army Weather Information Network (JAAWIN) Internet site. Due to multiple factors, including an increase in operations tempo and the improved resolution of meteorological models, the demand for access to customized aviation weather products has increased exponentially over the last few years. This, combined with the need for increased network security precautions to prevent unauthorized access to US Air Force weather data, has slowed the delivery of products. In its current configuration, JAAWIN provides its user-customized aviation weather products from a single Internet web site that has become a sort of "bottleneck" and a single point of

failure in the delivery system for this specific type of meteorological product. Up to this point, GrADS had served the Air Force meteorological community well in its role as the JAAWIN product generator for user-customized aviation weather products. The demise of GrADS use at the Air Force Weather Agency Technology Exploitation Branch is in sight however. The Air Force is restricted in what it can do to the GrADS software due to license issues. The Technology Exploitation Branch eventually wants to greatly reduce its dependence on GrADS and commercial-off-the-shelf (COTS) software applications (Telfeyan, 2001). As IGrADS was being readied to go operational on the Joint Air Force and Army Weather Information Network (or JAAWIN), the search had already begun for a stand-alone platform-independent solution that would eventually eliminate the Air Force's dependence on COTS software and GrADS.

In a 1997 proposal to the National Science Foundation, after 12 years of successfully supporting universities in the exploitation of information technologies in atmospheric research, the University Corporation for Atmospheric Research (UCAR) Unidata Program Center proposed a shift in its software emphasis to the JavaTM programming language. UCAR stated that the reason for the change of emphasis was to enable the Unidata academic research community to fully exploit the power of networking and distributed computing available through JavaTM. This shift in emphasis did not merely represent a change in programming language but rather a shift towards a new model of architecture-neutral, object-oriented, distributed, secure, multithreaded, and reusable software. Unidata emphasized that the advances achieved by JavaTM were "... so important that the greatest risk would be not to pursue them..." (UCAR, 1997) Born out of the proposal was a suite of applications known as MetApps (Meteorological

Applications). Over the past four years the UCAR Unidata Program Center has coordinated the further development of MetApps. MetApps is now a platform-independent meteorological software suite consisting of a set of object-oriented JavaTM classes and abstract data types (Caron, 1999). The question at hand is whether the Unidata MetApps classes and prototype applications are suitable for incorporation in an operational platform-independent interactive aviation weather products generation software package.

1.2 Statement of the Problem

There has arisen within the military meteorological community a need for multi-purpose interactive aviation weather products generation software solution. This software solution must satisfy six main requirements. First, the solution in question should be platform-independent. Due to the unpredictable nature of worldwide military operations, identical software platforms at diverse geographically separated locations cannot and should not be expected. Second, the software solution must not be restricted to obtaining data from a single source, as has been the case in the past. A single data source for meteorological data creates an opportunity for one's adversaries to undermine the capability of commanders to obtain the most current aviation weather information. Third, the solution must be robust. "Normal"-sized model data files should be able to be run on a component-based application without putting undue strain on a mid-level Windows PC or an average UNIX workstation. Fourth, the solution must be extensible or upgradeable. An application built using reusable components should be relatively easy to adapt to changes in its components (this requirement is somewhat subjective). Fifth,

the software solution should be user-friendly. Although this requirement/criteria is somewhat subjective, it must be kept in mind that users naturally will avoid software that is difficult to use. Sixth, the selected software solution must be an improvement over current visualization applications used in the operational aviation weather environment.

1.3 Scope

The Air Force Weather Agency Technology Exploitation Branch would like to know if Unidata Meteorological Applications (MetApps) stand-alone prototype applications or libraries are suitable for incorporation into next-generation platform independent interactive user-customized aviation weather products generation software. This project focuses specifically on the suitability of the MetApps prototype applications themselves and the component library. The scope of this work concentrates on exploiting the MetApps stand-alone prototype applications and library to their fullest extent through the construction of two experimental applications. In order to do this in the most efficient manner, data issues (such as format) had to be addressed first. It was only then that the remainder of this thesis could concentrate on the development of two platform-independent multi-use applications that fully exploit the capabilities of Unidata MetApps.

1.4 Overview of Approach

The overall approach taken in this thesis was one of software reuse and component-based engineering. Component-based software engineering (or CBSE) is a process that places emphasis on the design and construction of computing systems using reusable software components (Pressman, 2000). The CBSE approach can be illustrated

by thinking of it in terms of assembling a home stereo system from stereo components. Assembly of a system is easy because the user is not expected to build the system from a sundry mix of transistors, capacitors, and resistors. The stereo user may purchase his receiver, turntable, and cassette deck from different manufacturers and assemble them at home and be reasonably confident that the parts will work together. Component-based software engineering attempts to accomplish this same result with software. In this thesis, the “components” in question are the MetApps stand-alone prototypes and the classes contained in the MetApps library. The impetus behind a component-based and re-use approach is of course monetary. Just as it is cheaper to purchase stereo components rather than to design and build them, it is cheaper to reuse software components than to actually write the components oneself.

The analysis and design phase of this thesis started with research into the background of software systems that are currently used to deliver user-customized aviation weather products to customers worldwide. Research was also conducted into what already has been done in the area of component-based software development in the field of meteorology, specifically the use of the JavaTM programming language. The question of how the meteorological forecast model data should be handled, what data-format should be the standard, and where the data should be produced was addressed through analysis of several options. Among these was the feasibility of raw-data conversion by the user and the Abstract Data Distribution Environment (ADDE) client-server type architecture to obtain data. Next, copies of IMAST and IGrADS software were obtained from AFWA and analyzed with particular attention paid as to how IGrADS is designed. Following this step, analysis of how Unidata MetApps is structured

was conducted after a preliminary review of JavaTM syntax. The analysis phase was completed by constructing a tentative class diagram and system design of viable thesis prototype interactive software suite using TogetherSoft software. The second stage was the experimental or coding stage. During this stage two experimental approaches were implemented through the construction of two separate applications. The first of these applications was implemented using the Facade software design pattern. It used existing MetApps stand-alone prototype applications. The second application followed a hybrid Facade/adaptor software design pattern, exploiting the capabilities of the MetApps component library. The final stage of this research consisted of testing the experimental applications using available netCDF format meteorological model data files available locally and from the Internet using both UNIX workstations and Windows PC's. The experiments provided valuable information regarding the suitability of MetApps for incorporation in operational aviation weather products generation software.

1.5 Organizational Overview

Chapter two consists of an overview and discussion of related work in the area of weather data visualization tools. The tools discussed are not just those used by the military, but also those used in the academic community. The discussion begins with one of the older visualization tools, McIDAS, and ends with what could be described as one of the most cutting-edge tools, JavaTM-based 4DWX. Chapter three describes the experimental methodology used in the preparation of this thesis. This includes details on the design of the experimental programming environment and of the applications developed using the MetApps stand-alone prototypes and component library. Chapter

four discusses exclusively the results of this thesis research. Finally, Chapter five provides some conclusions and makes recommendations for future research. Included at the end of this document, as Appendix A, is a list of common acronyms used in this thesis.

2. Related Work

2.1 Overview

There has been widespread use of various meteorological data visualization tools throughout the worldwide meteorological community. Among these tools the most well known is McIDAS, the Man-computer Interactive Data Access System, which is supported by Unidata. Other tools used to construct customized meteorological products include the primary geophysical dataset visualization tools currently employed by the US Air Force. These are GrADS (Grid Analysis and Display System) and Vis5D, a system for interactive visualization of large five-dimensional (5D) gridded data sets. On the cutting edge of the development of geophysical data visualization tools are projects such as VisAD (Visualization for Algorithm Development) JavaTM component library and MetApps, which employs the VisAD library heavily. Lastly, building on the work done with VisAD is the US Army's 4DWX (Four-dimensional Weather) system, which is currently under development.

2.2 McIDAS

Developed in the 1970's, to fill a then void in visualization tools, the Man-computer Interactive Data Access System (McIDAS) was revolutionary in that it provided the capability of overlaying data from diverse sources. For example, surface temperatures could relatively easily be overlaid on a satellite image. An example of such a product is shown in Figure 1 below. McIDAS was also the first tool to allow the user to actually display meteorological information using colors and animation. McIDAS has

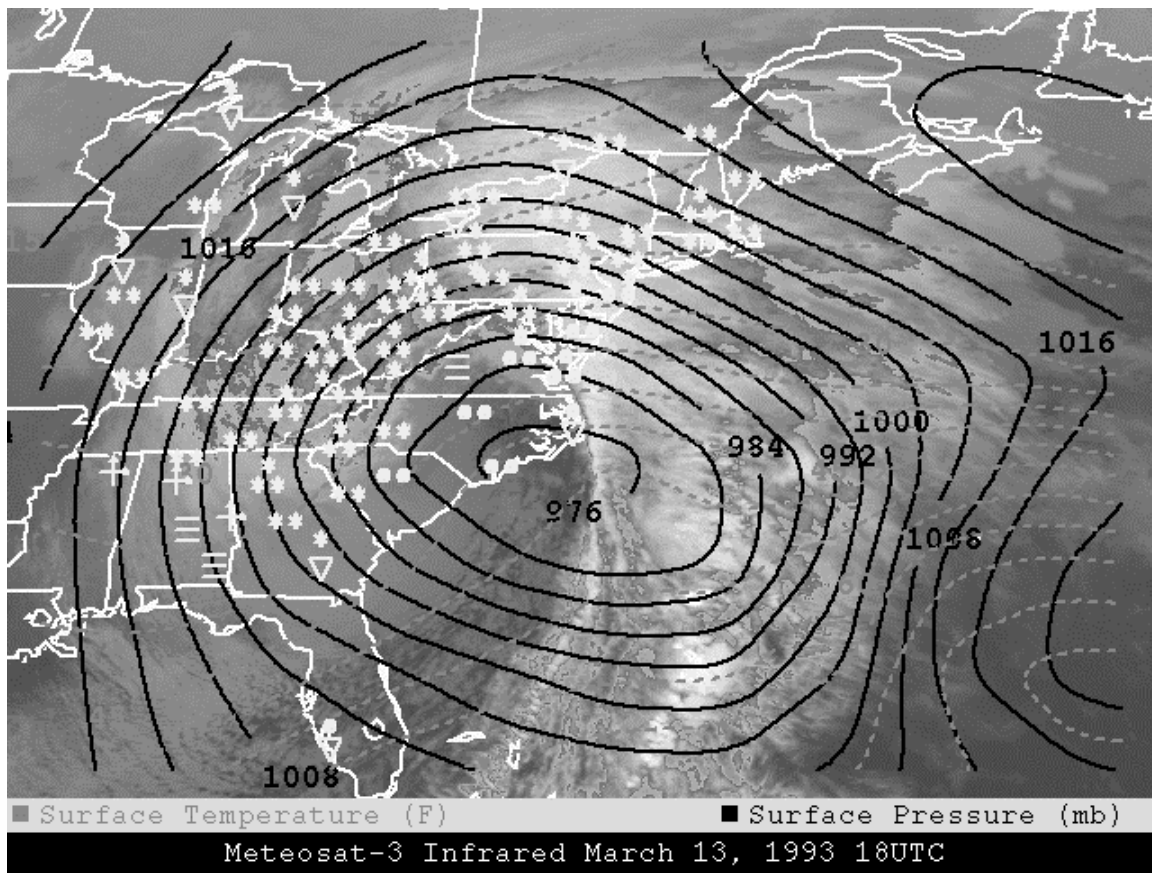


Figure 1. McIDAS Product. In this 13 March 1993 (Blizzard of '93) image, McIDAS was used to overlay temperature contours and weather data plots on top of enhanced satellite imagery (Image courtesy of SSEC, 1997).

been in use, and under continual development by the Space Science and Engineering Center (SSEC) of the University of Wisconsin-Madison since 1972. The McIDAS of today is a large, research quality, suite of applications used for decoding, analyzing, and displaying meteorological data for research and education. The software can be used with conventional observational, satellite, and grid-point data.

Unidata's distribution of McIDAS (a superset of SSEC McIDAS) has been under development since 1985 and in distribution since 1988. Unidata distributes a version of McIDAS known as McIDAS-X (the "X" implying a requirement for X-Windows), which

runs on a variety of UNIX platforms (UCAR, 2001). Many, if not all, later meteorological visualization tools, such as GrADS and Vis5D, owe much to the groundbreaking work done with McIDAS. McIDAS was the first so-called truly “user-friendly” visual meteorological application.

One of the benefits of McIDAS is that it has a long history. McIDAS has been around in some incarnation since 1972 and there is ample documentation for virtually everything that can possibly be done with the McIDAS software suite. It is by all standards a true classic in the field of meteorological visualization. There are however many drawbacks to McIDAS. First, it is not small. McIDAS consists of over one and one-half million lines of code contained in nearly four thousand modules (SSEC, 1997). It can occupy as much as one gigabyte of hard drive space not including any meteorological data files (which could add an additional gigabyte or two to the hard drive space occupied). Second, McIDAS only runs on specific UNIX systems running X-Windows and is thus not platform-independent. Third, McIDAS is not self-contained. As a prerequisite for using McIDAS, a variety of sundry software applications and libraries must be pre-installed in order for the McIDAS software suite to operate correctly. Last, even though McIDAS has been around since 1972, it is not bug-free. One of the more annoying of these bugs is the fact that the remote serving of meteorological sounding data does not work reliably under newer versions of the Linux operating system.

2.3 GrADS

The Grid Analysis and Display System (GrADS), developed by Mr. Brian Doty in conjunction with the Center for Ocean-Land-Atmosphere (COLA) Studies, is an interactive desktop visualization tool that is used for easy access, manipulation, and visualization of geophysical data. The GrADS visualization tool has been implemented worldwide on a variety of commonly used operating systems and is freely distributed over the Internet. The standard way of using GrADS is by executing operations interactively by entering FORTRAN-like expressions on a command line. GrADS does, however, have a programmable interface in the form of a scripting language, which allows for analysis and display applications. (COLA, 2001) GrADS scripts may display widgets as well as graphics. One of the common ways of exploiting this GrADS feature is by using Athena widgets to build a GrADS GUI (Graphical User Interface). Examples of how to use GrADS in this manner can be found on the NASA Goddard Space Flight Center, Data Assimilation Office Web pages (DAO, 2001). The GrADS scripting language can also be used to automate complex multi-step calculations or displays (COLA, 2001). In the case of the Air Force Weather Agency's Joint Air Force and Army Weather Information Network (JAAWIN), GrADS has been used to produce visualizations. These visualizations were then piped via the Interactive Meteorogram and Skew-T (IMaST) system back to the user who requested the product. IMaST is actually an ingenious JavaScript interactive interface developed by the Air Force Weather Agency Technology Exploitation Branch for GrADS. It allows a user to produce a meteorogram, such as the one shown in Figure 2 below, or skew-T diagram (both specialized visualizations) for any location within the domain of a user-determined meteorological

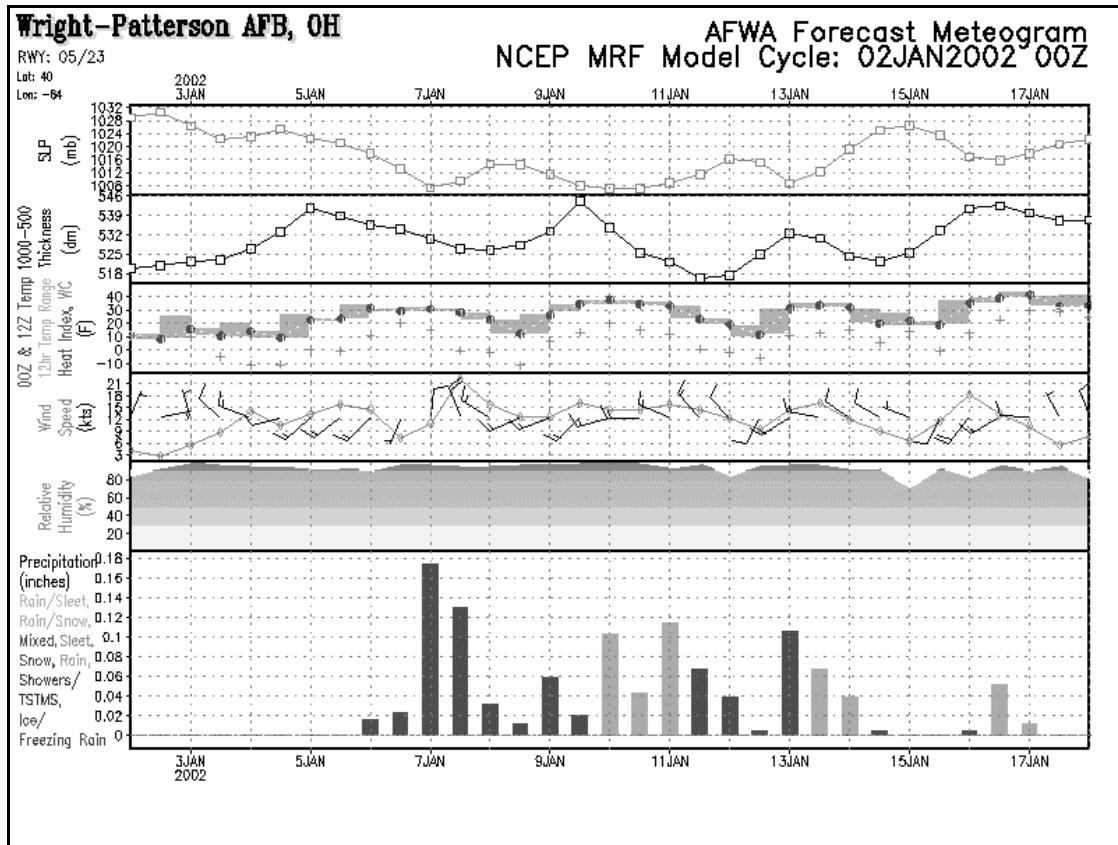


Figure 2. GrADS Meteorogram. Depicted here is a meteorogram produced using GrADS and the IMAST interactive user interface. These products can be custom built using a customer's meteorological model and location specifications (Meteorogram courtesy of Air Force Weather Agency).

forecast model. The user can enter either map coordinates, a 4-letter ICAO (International Civil Aviation Organization) identifier, or pick a location on a number of theater maps.

On the positive side, GrADS is much smaller than McIDAS and a Win32 port of GrADS has been developed that can be run on a Windows PC. The drawback of this Win32 port is that one must have an X-server running in order to display graphics. There is also a Mac version of GrADS in existence but neither COLA nor any other entity supports it. GrADS is also more portable than the McIDAS application suite requiring a

user to download only three files (besides data files). It also has the powerful capability of reading GRIB formatted data (one of the more popular meteorological data formats) directly. Finally, GrADS is well documented and widely used among meteorologists. On the down side, GrADS is copyrighted software, and although the source code is available, it is subject to restrictions. In addition, GrADS, although a robust application, is not platform independent. GrADS requires an X-windows server to function correctly.

2.4 Vis5D

Vis5D, used widely by the Air Force Combat Climatology Center, is a software system that can be used to visualize both gridded data and irregularly located data. Sources for this data can come from numerical weather models, surface observations and other similar sources. Vis5D is designed to work with data in the form of a five-dimensional (5D) rectangle. In other words, the data consists of real numbers at each point on a "grid" which spans three space dimensions, one time dimension, and a dimension for enumerating multiple physical variables. Vis5D does not actually "require" the use of a 5D rectangle. This software will work on data sets with only one variable, one time step (i.e. no time dynamics) or one vertical level. Vis5D can also work with irregularly spaced data, which are stored as "records". Each record contains a geographic location, a time, and a set of variables, which can contain either character or numerical data.

A major feature of Vis5D is support for comparing multiple data sets. This extra data can be incorporated at run-time as a list of "v5d" files or imported at anytime after Vis5D is running. Data can be overlaid in the same 3-D display and/or viewed side-by-

side spreadsheet style. In the spreadsheet style, multiple displays can be linked. Once linked, the time steps from all data sets are merged and the controls of the linked displays are synchronized. An example of such a spreadsheet display is shown in Figure 3 below. The Vis5D system includes the vis5d visualization program, several programs for managing and analyzing five-dimensional data grids, and instructions and sample source code for converting data into the native Vis5D file format (SSEC, 2000). Vis5D visualizations have the distinction of appearing almost identical to VisAD visualizations, VisAD being one of the required sub-components for MetApps.

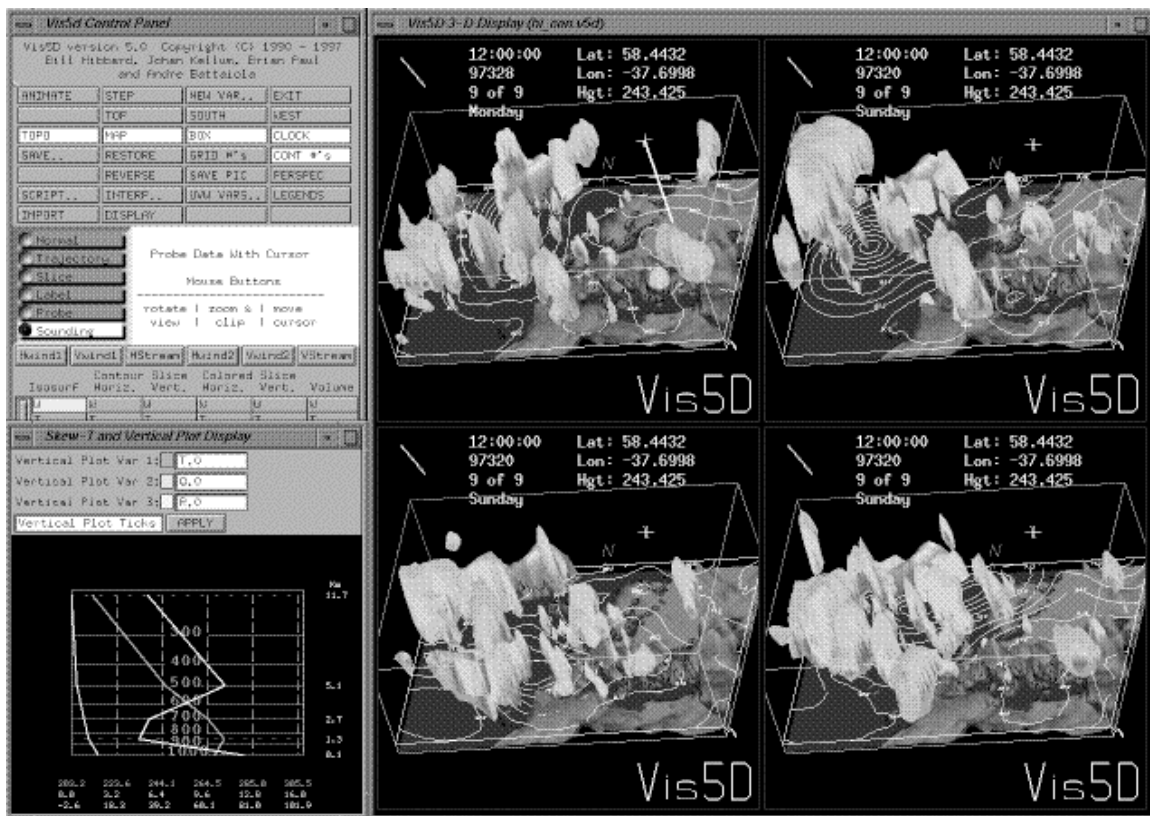


Figure 3. Vis5D Screenshot. This screenshot shows Vis5D generating a spreadsheet display of four members of an ECMWF ensemble forecast (Image courtesy of SSEC, 1998).

Another important feature of Vis5D is the Vis5D API (Application Programmer's Interface). This API is an interface between the Vis5D user interface and the Vis5D core software. This API allows developers to include Vis5D as a visualization subsystem of other systems, where the other system invokes the Vis5D core through the API. Thus one can incorporate the functionality of Vis5D into the user interface of a primary system without abandoning the user-interface style of that primary system.

Among the drawbacks of Vis5D is the fact that, like GrADS, it is protected by copyrights. It is also not platform-independent, the code having been written in the C (like GrADS) and the FORTRAN programming languages. Finally, Vis5D development has ceased at the Space Science and Engineering Center (SSEC) even though the latest version of the software is available on the SourceForge Vis5d+ Internet site (SourceForge, 2001). SSEC has chosen instead to concentrate its resources on the further development of VisAD, the JavaTM incarnation of Vis5D.

2.5 VisAD

The word "VisAD" is an acronym for "Visualization for Algorithm Development". VisAD is a JavaTM component library (consisting of a package of JavaTM classes) providing support for interactive and collaborative visualization and analysis of numerical scientific (especially geophysical) data. The library was created and is actively being developed by the Space Science and Engineering Center (SSEC) at the University of Wisconsin. VisAD provides the building blocks (or components) for application developers to more easily create scientific applications using the JavaTM programming language. The VisAD library, however, is not and was never intended to be a stand-alone

application. Nevertheless, included with VisAD are a number of experimental applications that exploit the features of its component library. Among these is the VisAD *SpreadSheet* application (a screenshot of which is shown in Figure 4).

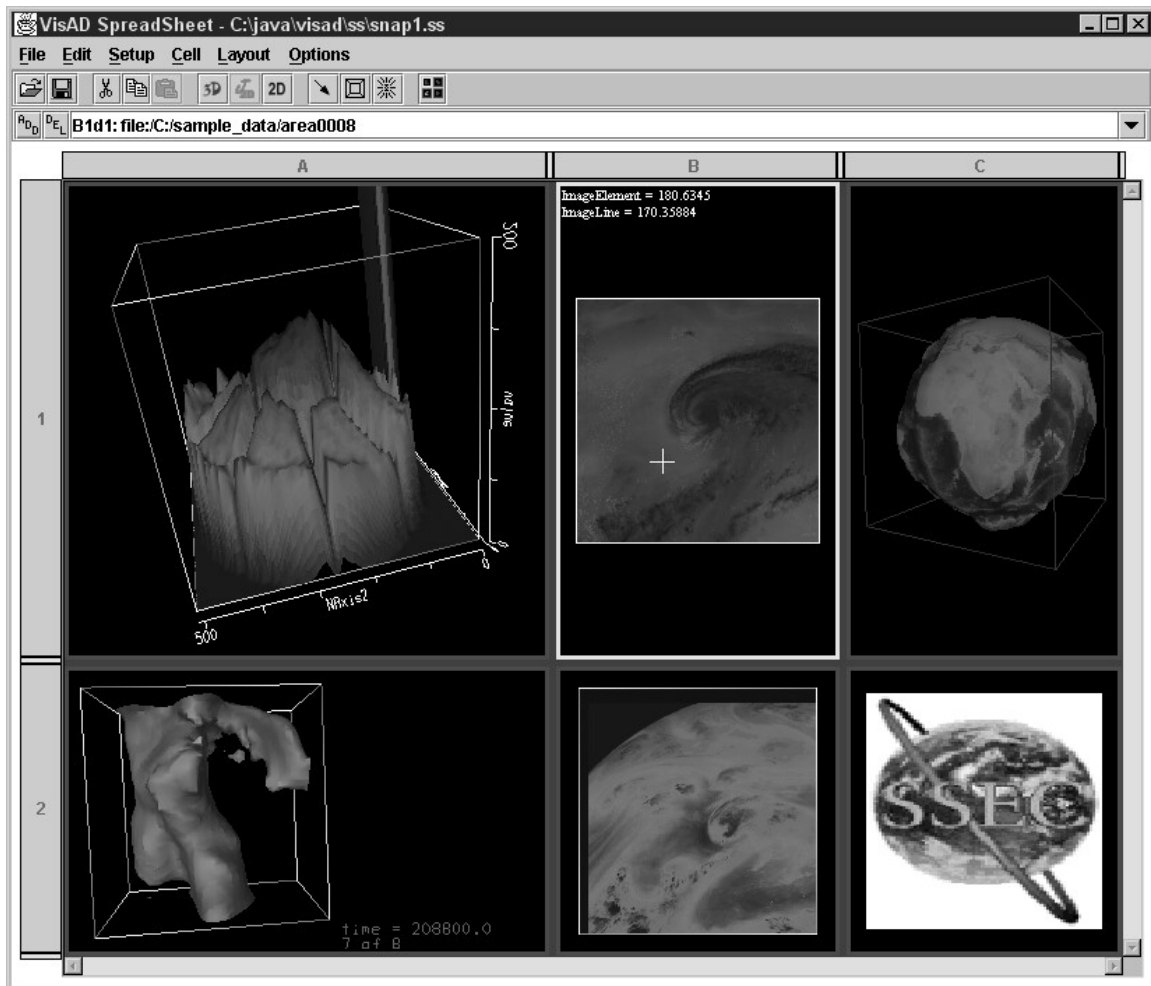


Figure 4. VisAD Screenshot. This screenshot shows a visualization spreadsheet application built using the VisAD library. Of note are the similarities in appearance to Vis5D imagery and color schema (Image courtesy of SSEC, 2001).

VisAD has become the backbone of the MetApps and 4DWX visualization systems (both discussed later in this chapter). The question is why. Why would the

developers of MetApps and the Army's 4DWX systems select the VisAD component library to visualize their meteorological model data? There are actually four important factors (or reasons) that lead to the natural selection of VisAD as part of a visualization solution.

The first factor is the VisAD library's platform independence and open source licensing. The VisAD system uses pure JavaTM for platform independence and to support data sharing and real-time collaboration among geographically distributed users. This fits in well with the goals of many current and future JavaTM software development projects and makes way for interactivity via the Web. By necessity, JavaTM code is object-oriented thus promoting extensibility and code reuse, which drives down the cost of development. In addition, the VisAD code is open source Lesser GNU Public License (LGPL). Anyone can use the code or contribute to it. As a result there is a very supportive community that correct problems and add features to the library regularly.

The second factor has to do with the VisAD data model. VisAD incorporates a general mathematical data model that can be adapted to virtually any set of numerical data, which supports data sharing among different users, different data sources, and different scientific disciplines. This provides access to data that is independent of storage-format and location. The VisAD mathematical data model has been adapted to many file formats one of which is netCDF (network Common Data Form), which is the primary data format used by MetApps. Another aspect of the data model is internal representation of data. One particular data set might have a one-kilometer resolution while another has a four-kilometer resolution. One domain might be defined by equal spacing from a given location while another has an irregular longitude-latitude grid. One

sensor might provide temperature in Celsius while another reports in Fahrenheit. One model might provide upper air data on pressure levels while another provides height above ground. Designing and implementing a software data structure to deal with these disparities is a monumental task. VisAD provides a solution to the problem of designing and implementing a software data structure to deal with data differences. It provides a sophisticated data model that allows an application developer to represent many different types of data in a consistent manner. (Lindholm, 2001)

The third factor is the VisAD display system. VisAD utilizes a general display model that supports three-dimensional (3D) interactivity, data fusion, multiple data views, direct manipulation, collaboration, and virtual reality. VisAD offers the application developer a simple interface for displaying data while hiding the complexities of creating graphics. Data can be displayed as lines, points, images, contours, or customized shapes. The software developer also has control of display properties such as data ranges, color, and line width. Animation support is also included. A 3D view can be obtained by mapping a field (such as altitude) to the Z-Axis. The user can rotate, zoom, and pan the scene. VisAD also provides a direct manipulation mechanism. A user can "grab" the data on the display with the mouse and move or redraw it. As a result, the underlying data object's data values will be changed. (Lindholm, 2001)

The fourth factor is that support for two distinct communities is built-in: developers who create domain-specific systems based on VisAD, and users of those domain-specific systems. Because of the way VisAD is designed, it supports a wide variety of user interfaces, ranging from simple data browser applets to complex

applications that allow groups of scientists to collaboratively develop data analysis algorithms. VisAD provides for developer extensibility in as many ways as possible.

All these factors lead one to the unmistakable conclusion that VisAD provides an abundance of data visualization and analysis functionality greatly enhancing an application developer's ability to create powerful scientific applications while affording the flexibility for customization.

2.6 MetApps

In academia, as previously mentioned, the University Corporation for Atmospheric Research (UCAR) Unidata Program Center has over the past four years coordinated the development of Meteorological Applications (MetApps), a platform-independent meteorological software suite and visualization tool consisting of a set of object-oriented JavaTM classes and abstract data types (Caron, 1999). The goal of the MetApps project is for MetApps to operate both as a stand-alone application (or set of applications) and as part of larger distributed client server architectures that enable growth in both functionality and the number of user workstations (UCAR, 2000). MetApps software is freely available and is protected under the Lesser GNU Public License (LGPL), which means that MetApps can be tailored to meet individual programmers' needs. The MetApps prototype applications that have been developed up to this point are designed to run on any platforms that fully support JavaTM 2 and Java 3DTM. Currently, the MetApps prototypes are being tested mainly on the Windows, Solaris, and Linux platforms. Shown in Figure 5 is a screenshot of the MetApps Gridded Data Viewer (GDV), the most recently developed prototype application. Amply evident

is the use of VisAD library display components to generate an interactive meteorological product. The VisAD Java™ library is required for most of the MetApps components.

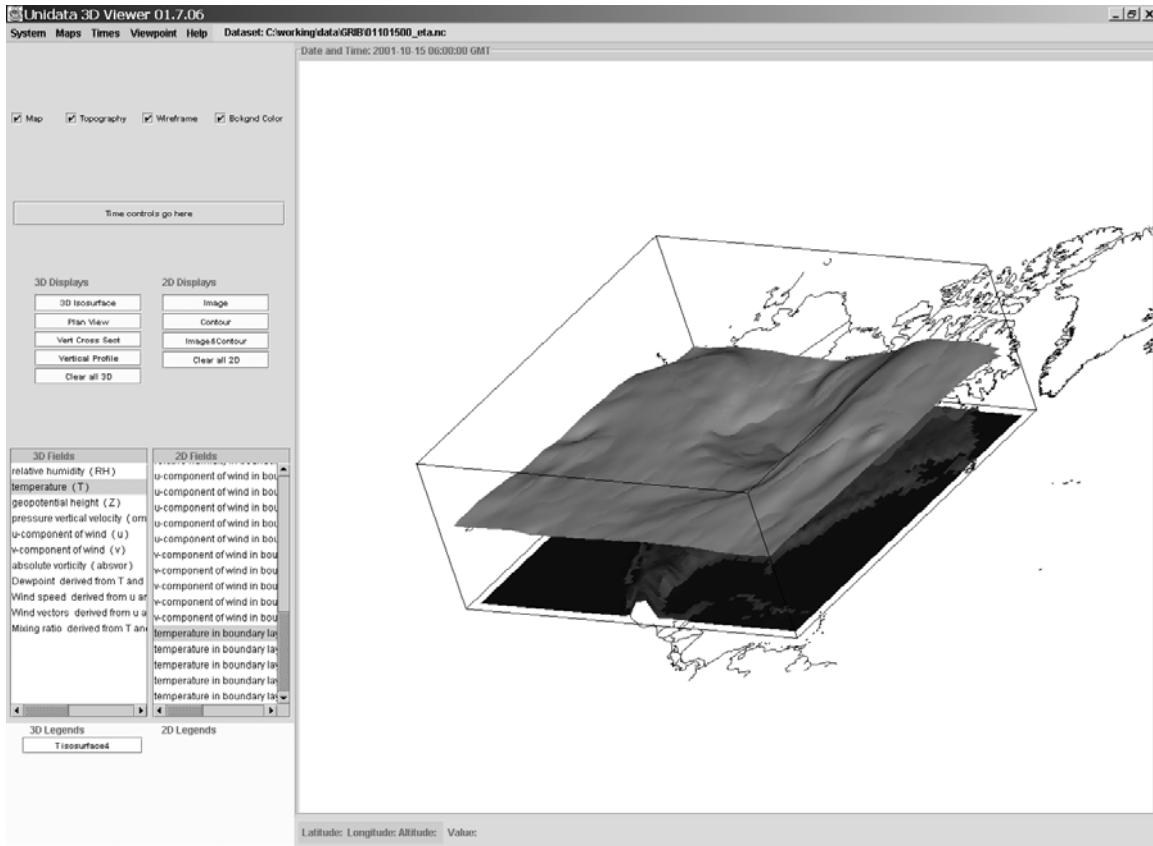


Figure 5. MetApps Screenshot. This screenshot shows the MetApps 3D Gridded Data Viewer (GDV) showing -25°C temperature isosurface. The use of VisAD library components should be apparent in the creation of the image portion of the display (refer to Figure 4).

MetApps is a work in progress and part of the ongoing research sponsored by Unidata. This means that the classes contained in the MetApps library are in a state of flux. Changes and/or “improvements” to the library should be expected. While some parts of it have stabilized for now (such as netCDF for Java™) others remain volatile

(such as VisAD, which is undergoing continual development). A more extensive analysis of the design and capabilities of the MetApps prototype applications and library are provided in Appendix B of this thesis. The development of MetApps has not gone unnoticed. Because of the lower costs involved in reusing software components to develop applications, much interest has been shown in MetApps by developers outside of the academic research community. In this thesis, efforts concentrate on determining the feasibility of MetApps for incorporation into a system for providing user-customized aviation weather products in an operational rather than in a research environment.

2.7 4DWX

Back in the military community, completely independent from current Air Force Weather Agency efforts, research has been conducted into a Java™ implementation of meteorological visualization tools under the acronym 4DWX. The concept for the Four-Dimensional Weather (4DWX) System originated in the recommendations of a study commissioned in 1995 by the Atmospheric Sciences Division (now Atmospheric Sciences Branch) at Headquarters, U.S. Army Test and Evaluation Command (ATEC). At the time, the Atmospheric Sciences Branch recognized that the U. S. Army Test and Evaluation Command meteorological support function had to change its way of doing business if it was to help meet the challenge of testing new and more complex materiel, such as smart munitions, at a time of declining resources for test and evaluation. The National Center for Atmospheric Research (NCAR) found that the Army Test and Evaluation Command was not making full use of its meteorological information because of problems in four areas: data base management, data assimilation, modeling, and user

displays (Bowers, 2000). NCAR also recommended a set of solutions to the identified meteorological support problems. Promoting the concept of software reuse, wherever possible, the recommended solutions were based on existing software, including the NCAR/Penn State non-hydrostatic Mesoscale Model Version 5 (MM5), the Zebra data archival and display system (developed by the NCAR Research Data Program), and the thunderstorm AutoNowcaster algorithm. The Atmospheric Sciences Branch decided to proceed in Fiscal Year 1997 with the development of a prototype 4DWX system to implement NCAR's recommendations. (ATEC, 2001)

As part of 4DWX, the Visual Meteorology Tool (VMET) was developed as the versatile display component of the 4DWX system. It was designed to enhance the forecast and research meteorologist's ability to visualize, explore, manipulate, and analyze meteorological model data sets in two, three, and four dimensions. VMET can also serve as a virtual environment in which to run virtual experiments in real weather. In addition, VMET is written entirely in JavaTM and built upon the VisAD libraries. Lastly, VMET can integrate a variety of data sets into a single interactive two or three-dimensional display (NCAR, 2001). A screenshot of VMET is shown in Figure 6. It is apparent from the screenshot that like the MetApps Gridded Data Viewer, VMET owes a lot of its visualization functionality to the VisAD library.

The 4DWX program, however, was not without its problems. The program included a groundbreaking attempt at using a database to manage most of the primary data in a major operational system. At the start of the program, the plan was to apply the latest database technologies to all of the data sets, including the large 3D output volumes generated by the MM5 meteorological forecast models. After a period of extensive

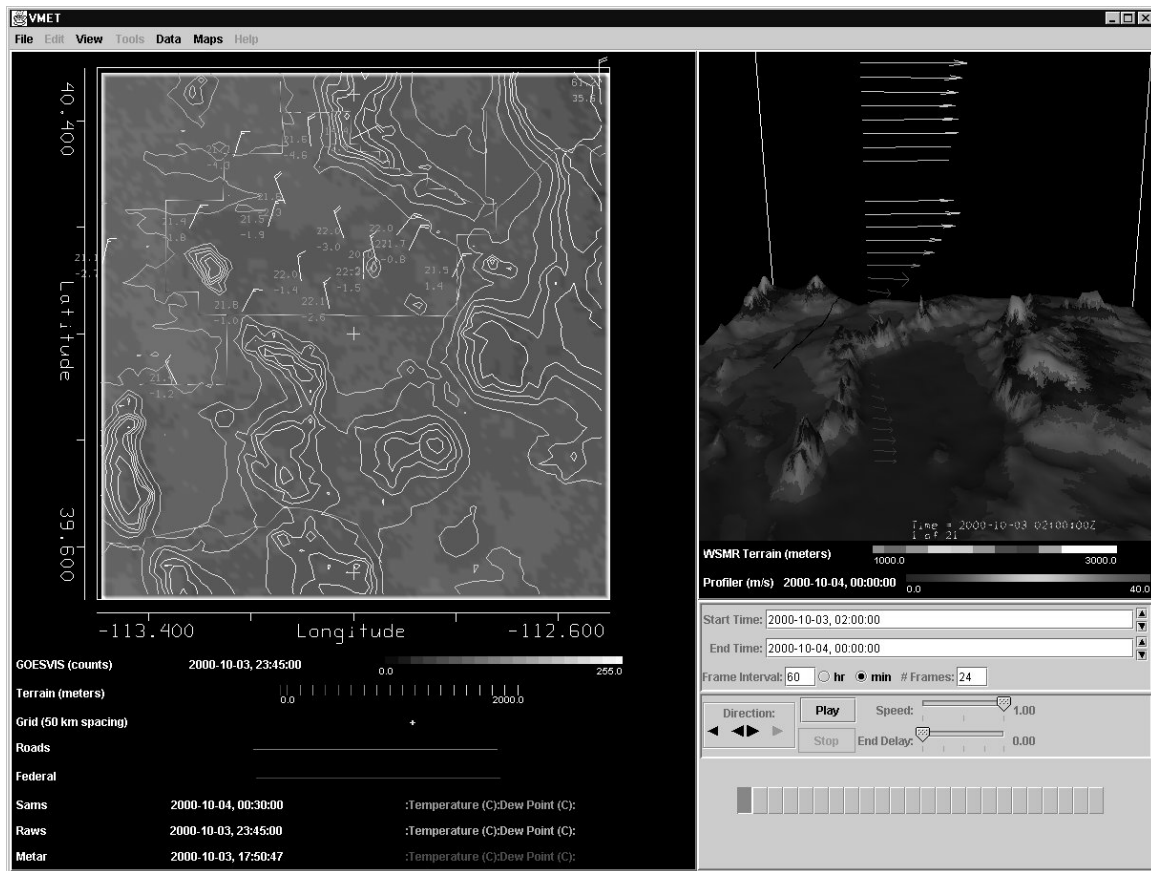


Figure 6. 4DWX VMET Screenshot. This screenshot shows an example of an application composed of multiple 4DWX VMET components combined with Java™ Swing components. This application bears many similarities to the MetApps Gridded Data Viewer prototype application shown in Figure 5, a direct result of the wide use of the VisAD component library (Image courtesy of UCAR, 2001).

testing, it was determined that performance issues and implementation complexity called for a scaled-down plan for the database, restricting it to the collection of measured observation data sets needed in the 4DWX system (NCAR, 2000).

2.8 Summation

In the end, the very existence of MetApps and 4DWX is important to this thesis research. The development of projects such as MetApps in the civilian and 4DWX in the

military community shows that the cutting edge developments in meteorological visualization almost exclusively rely on component-based solutions and the JavaTM programming language. Today more applications than ever must interface with the Internet, and JavaTM is the natural solution. Apart from being the preeminent language of the Internet, JavaTM is important for other reasons not the least of which is that it has altered the way in which we program computers. JavaTM has come a long way from its early days as a platform-independent language that could be used to create software for embedding in consumer products such as microwave ovens. The Internet, with its dynamic environment and wide variety of different computing platforms, helped catapult JavaTM to the forefront of programming today making it in many cases the language of choice. Compared to the other mainstream computer languages of today, such as C, C++, and FORTRAN, it is only in speed and performance where JavaTM lags behind. But even here, JavaTM is catching up rapidly (Schatzman, 2001).

3. Methodology

3.1 Overview

This chapter provides a detailed description of the approaches used in accomplishing this thesis. A description of the high and low-level goals and the criteria for success are provided. Next is a description of the experimental software and hardware environments for the experiments performed in this thesis. These environment descriptions are followed by an explanation of two separate experimental approaches. The first approach depends on the stand-alone MetApps applications while the second utilizes the MetApps library. Rounding off this chapter is a section dealing with the important issue of data requirements and handling. This last section is important since a measure of success of this project can be measured by how well Unidata MetApps deals with the meteorological data issues.

3.2 Goals

The guiding goal for this thesis is to find a multi-purpose interactive military aviation weather product generation software solution. This software solution must be platform-independent, multiple data source access configurable, robust, extensible or upgradeable, user-friendly, and an improvement over current visualization applications used in the operational military aviation weather community. Another goal is to determine if a software reuse and component library-based approach can be used in such a solution. A sub goal of this is to determine whether the JavaTM-based Unidata MetApps components are suitable for fulfilling this software component library role. A final sub

goal is to provide a realistic assessment that can be used to determine if Department of Defense (DoD) funds should be diverted into the development of meteorological visualization software based on Unidata MetApps rather than in-house DoD projects.

3.3 Requirements for Success

The requirements for success of the software solution are six-fold and largely objective. The first of these requirements is that the software solution developed using the existing components be platform-independent. Due to the unpredictable nature of worldwide military operations, identical software platforms at diverse geographically separated locations cannot and should not be expected. In a deployed situation, military weather personnel cannot be guaranteed a particular “standard” computing environment. The results achieved using the selected software solution on a Sun Solaris workstation should be virtually identical to those achieved on a Windows PC. Second, the software solution must not be restricted to obtaining data from a single source, as has been the case in the past. A single data source for meteorological data creates an opportunity for one’s adversaries to undermine the capability of commanders to obtain the most current aviation weather information. Multiple configuration options on a selected software solution should effectively mitigate or eliminate this problem. Third, the solution must be robust. 35 megabyte (MB) Nested Grid Model (NGM) Grid in Binary (GRIB) files, 6 MB McIDAS AREA files, and 2 MB upper air files should be considered “normal”-sized data files. “Normal” data files should be able to be run on a component-based application without putting undue strain on a mid-level Windows PC or an average UNIX workstation. Most meteorological personnel have limited experience in dealing with

software problems. Under operational conditions they should not be required to constantly “fix” bugs in their software systems. Fourth, the solution must be extensible or upgradeable. The reason for this is cost. It is usually cheaper to make upgrades or adaptations rather than to replace an entire system. An application built using reusable components should be easier to upgrade and adapt since components can be selectively replaced. Fifth, the software solution should be user-friendly. Although this requirement/criteria is somewhat subjective, it must be kept in mind that users naturally will avoid software that is difficult to use. Military users should be capable of interactively generating their own user-customized aviation weather products upon demand. Sixth, the selected software solution must be an improvement over current visualization applications used in the operational aviation weather environment. This last requirement may prove difficult to measure or fulfill. The simplest way to make this determination is to make a subjective comparison between any new solution and the GrADS solution currently used by Air Force weather. Considerable operational experience has been built up over the years with applications such as GrADS. The GrADS software application has the capability to display time series and, through scripts, meteorograms. Most component libraries have yet to incorporate such functionality.

3.4 Design of Experimental Applications

3.4.1 Overview

The experimental approach taken in this thesis involves the construction of two experimental applications. These applications demonstrate and test the functionality of

the software components in question and determined their actual level of functionality. The premise of this thesis is that Unidata MetApps is suitable for fulfilling the role of the component library used in developing the operational application that fulfils the requirements set forth in the previous section. A two-pronged approach is taken based on the premise that there are actually two types of MetApps. The first MetApps is literally, as the acronym implies, composed of meteorological applications. The second MetApps is a JavaTM class library. Thus, the two experimental applications constructed in this research are based on the duality of the MetApps project. Both experiments use a component-based approach to the construction of the experimental applications. In traditional object-oriented programming the software reuse is primarily achieved through inheritance of classes, while in a component-based approach reuse is achieved through interfaces with independently developed software. Therefore, the use of a component-based approach does not preclude one from applying object-oriented principles. The component-based approach is similar to construction of computer system hardware from its constituent parts (a CPU from Intel, a soundcard from Creative, memory modules from Samsung, etc.) or to putting together a stereo system, as previously mentioned.

A component-based approach is chosen because whether VisAD, MetApps, or some other library is used, one is still essentially dealing with the assembly of components to build applications or systems. By concentrating on component architecture one can separate a system or application into independent subsystems. Particular subsystems can then be manipulated, changed, or adapted at the discretion of a developer with minimal impact on the system as a whole. Lastly, using a component-based approach on both of the experiments provides comparison and contrast. This gives

a method to determine whether the prototype applications or the library are better suited to the goals of this thesis.

3.4.2 Application Design Approach 1

The design approach chosen for the first experimental application resembles closely the Facade software design pattern. Figure 7 shows a much-simplified view of the Facade software design pattern. The purpose of the Facade pattern is to shield a software developer from the possibly large amount of recoding involved in an application due to interface changes in a subcomponent. An object's interface describes the complete set of requests that can be sent to that object. The Facade provides a unified interface to a

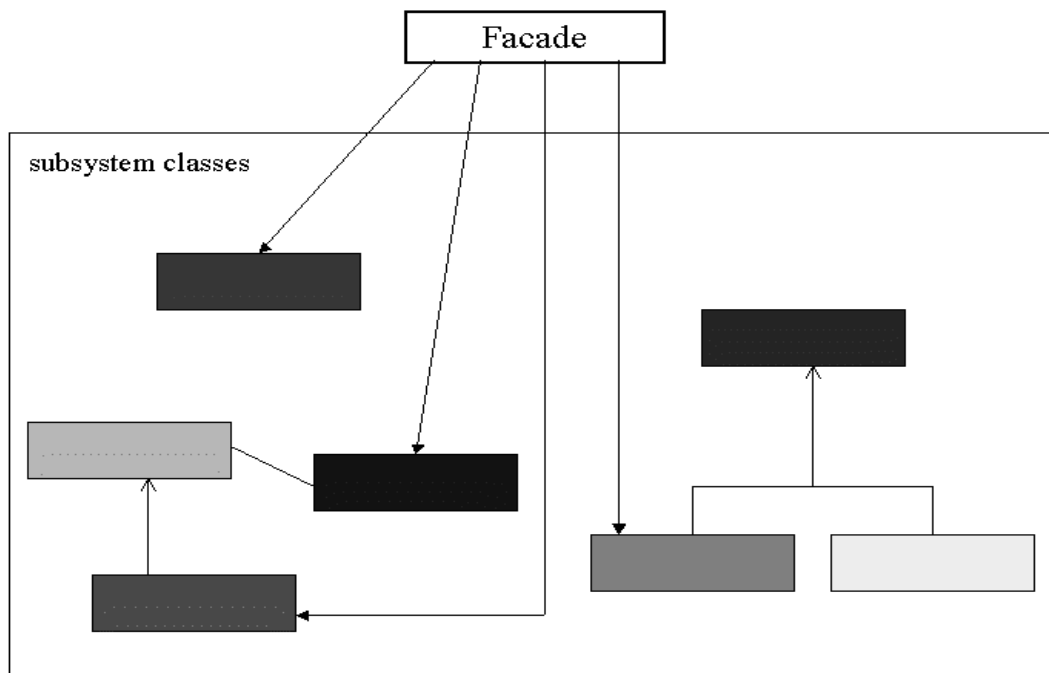


Figure 7. Standard, simplified view of the Facade software design pattern. Adapted from an original diagram by Shalloway and Trott (Shalloway and Trott, 2002).

set of interfaces. The motivation behind using this design pattern is that it helps reduce complexity and minimizes the task of tracking down every call to an interface. Using a Facade pattern promotes weak coupling (Gamma, 1995). Weak coupling makes it easier to replace or change subcomponents without disabling the complete system. The biggest drawback to using a Facade-type pattern is that it results in a system that is slower and harder to understand (Vlissides, 1996). The reason for this is that the Facade adds additional call levels to a software system.

The first experimental application in this research uses the existing stand-alone prototype applications contained in the component library, in this case MetApps. An interface is required that links the library's existing prototype applications together in a single application. This interface accesses the necessary compiled classes in the application libraries to achieve what appears as a seamless integration of the parts. In addition, a number of other library files are required for the applications to function properly. A complete listing of these required libraries is provided in the detailed description of MetApps contained in Appendix B. It should be pointed out that some prototype applications do not have any additional library requirements. The MetApps Interactive Sounding Application in this design approach as depicted in Figure 8, is completely self-contained and has no special requirements except a JavaTM runtime environment (JRE). The Interactive Sounding Application does not require the use of additional libraries because the classes it requires from those libraries are packaged together with the *sounding* classes. Each tabbed rectangular box in Figure 8 represents a library or package. The dashed arrows show how some libraries, or packages, depend on other libraries. For example, the Image Viewer Application depends on the *JavaHelpTM*,

VisAD, *MetApps*, and *Data Files* packages/libraries to function properly. The bold dashed enclosure shows what is required for the GDV while the bold dotted enclosure shows the Image Viewer requirements. It should be apparent from Figure 8 that the *MetApps* prototype applications are dependent on a core set of shared classes contained in a set of libraries.

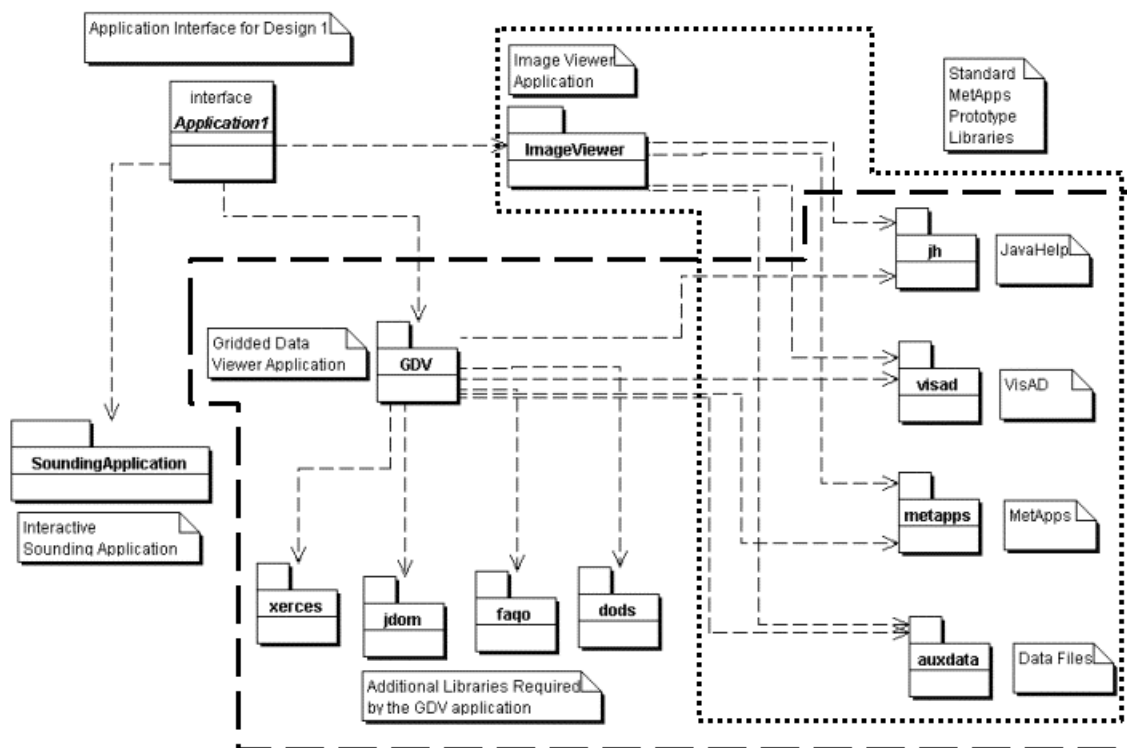


Figure 8. Application 1 Diagram. This figure shows a package diagram depicting the dependencies between the *MetApps* stand-alone applications and the required Java™ libraries and ancillary data files.

In the case of *MetApps*, the Facade-based approach has a number of advantages, but also some disadvantages. One advantage is that with the exception of the graphical user interface, it is expected that pre-compiled Java™ bytecode can be used without

significant problems. The interface would have to be crafted using some type of interactive development environment or IDE. Among the advantages is that there is less deprecation in stand-alone applications. Deprecation is what happens to functions when they are retired. When a function is deprecated, it means that the method was once valid, but has now been replaced by a newer method. Deprecation is a problem with MetApps because required external libraries, such as VisAD, are updated frequently. This problem is not unique to MetApps. It is a problem with most software projects that depend on components that have been developed independently from said project. The problem can be eliminated by copying the offending library into the actual application, thereby shielding the application from unforeseen changes in the component library. Another possible advantage of this particular design approach is the modularity of this design. In theory, when new stand-alone applications are created they can be added with minimal rewriting of existing application code. One of the drawbacks to this experimental application design is that it is difficult to make even small changes to the inner workings of the individual MetApps stand-alone applications unless the source code has been made available. These parts of the application are by definition “self-contained.”

3.4.3 Application Design Approach 2

The second experimental design approach is similar to the first. This approach somewhat resembles the Facade software design pattern described previously above. However, the design pattern more closely resembles a hybrid of both the Facade and Adapter software design patterns. The purpose of an Adapter pattern is to convert the

interface of a class into another interface. In other words, a new interface is constructed for an object that works correctly but has the “wrong” interface. In this approach the actual component class library is used rather than precompiled prototype applications. In this design an application is constructed using the base classes contained within the component library. Instead of an application interface, this design depends on an application class that depends on the component classes. Unidata has indicated that in order to use the MetApps library in this manner the additional libraries of JavaHelpTM, VisAD, and the Data Files must be included in any JavaTM project. A decision was made to not include the netCDF-2 JavaTM library and instead allow for dependence of the MetApps classes on the internal version of netCDF-2. This was a logical step since the internal version of netCDF-2 is virtually identical to the current official netCDF-2 JavaTM library. Preliminary analysis of the MetApps library in the course of this research indicates that there are a number of other libraries that are required for construction of MetApps-based applications. Table 3 below lists these additional libraries.

Table 1. Additional Libraries Required for Design Approach 2

Library	Filename
JNLP	jnlp.jar
HTTP Client	HTTPClient.jar
Data	data.jar
Units	units.jar

The *jnlp.jar* file provides support for Java™ Network Launch Protocol (JNLP). JNLP allows a user to run a Java™ application or applet directly from the Internet. JNLP provides direct access to Java™ software using the latest Java™ virtual machine (or JVM) without the constraints and problems of Java™ applets within web browsers. The *HTTPClient.jar* package provides a complete HTTP (Hypertext Transfer Protocol) client library. The *data.jar* file contains some additional ancillary files. Lastly, *units.jar* is actually a sub-package of the MetApps library. It is included in this experimental design to help determine if separating the sub-components of MetApps results in a more robust application. Figure 9 shows how the MetApps component library depends on all the different libraries described above. What is not shown in Figure 9 is that there are dependencies between these individual libraries as well (too numerous to depict).

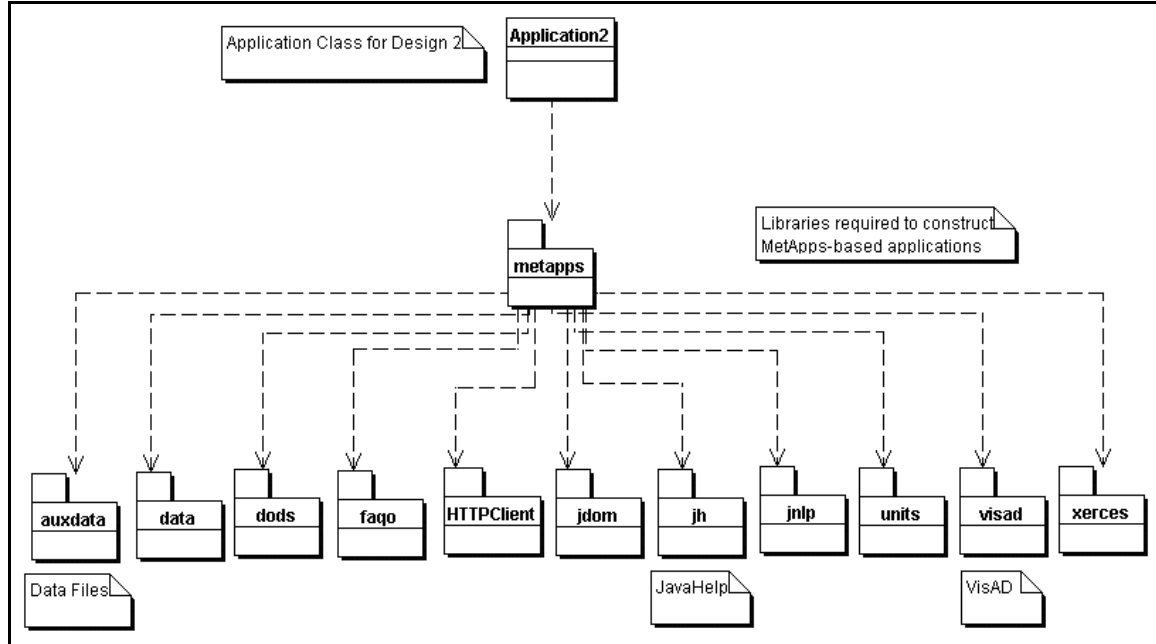


Figure 9. Application 2 Diagram. This figure depicts a mixed class/package diagram showing the relationships between the experimental application and libraries in building applications using Approach 2.

The end result of the second design approach should mimic the results achieved by the first design approach although it is suspected that this approach has greater stability. The reason for this increased stability is that in this approach we build an application using all source code rather than a combination of source and compiled bytecode. An additional benefit of this design is that a developer has greater control of how his or her application behaves. Minor changes to the code can easily be made without resorting to de-compilation of the original Unidata MetApps Java™ bytecode. One foreseeable drawback to this approach is the inevitable deprecation of some of the older component classes that depend so heavily on VisAD. Problems are foreseen with compilation of an Interactive Sounding Application since it depends so heavily on the VisAD library. The VisAD library has undergone considerable change since the original Interactive Sounding Application was released with an early version of VisAD. This problem with deprecation could possibly negate any increase in application stability achieved through the use of source code compilation.

3.5 Experimental Programming Environment Setup

Table 2 shows the setup of the experimental software environment for this thesis. MetApps, which is theorized to fulfill the criteria described in section 3 above, is designed to run on any platforms that fully support Java™ 2 and Java 3D™. Java™ 2 Platform, Standard Edition (J2SE) provides the foundation for the construction of the Java™ applications in this thesis. The J2SE is implemented by the Java™ 2 Software Development Kit (SDK), Standard Edition, and the Java™ 2 Runtime Environment

(JRE), Standard Edition. The J2SE is needed to run the MetApps applications. This thesis used the latest version of Java™ 2 from Sun Microsystems (version 1.3.1).

Table 2. Experimental Software Environment

Software
Java™ 2 Platform, Standard Edition (J2SE) <ul style="list-style-type: none"> - Software Development Kit (SDK) 1.3.1
Java 3D™ Software <ul style="list-style-type: none"> - Java 3D™ API version 1.2.1 (either OpenGL or DirectX version).
Various Sundry Packages <ul style="list-style-type: none"> - JavaHelp™: A full-featured, platform-independent, extensible help system that enables developers and authors to incorporate online help in applets, components, applications, operating systems, and devices. It is being used as the interface to the on-line documentation for the MetApps prototypes. - VisAD: A Java™ class library for interactive and collaborative visualization and analysis of numerical data. Its data model and display capabilities are being used in several of the MetApps components and prototypes. - MetApps: A standard library of classes developed by Unidata (in bytecode). - Ancillary Data Files: Ancillary data files (icons, enhancements, map files) used by the MetApps components and prototype applications.
Java™ Development Environment <ul style="list-style-type: none"> - Borland JBuilder 5 Professional: Visual Java™ 2 development environment.

Unidata recommends that Solaris users install a number of system patches for Java™ 2 to work correctly (UCAR, 2000). Preliminary experimentation with the MetApps prototypes, by the author of this thesis, indicates that these software patches are necessary for the stability of the MetApps applications and UNIX operating system.

In the same directory as Java™ 2, Java 3D™ must be installed on the system being used. The Java 3D™ API is a set of classes for writing 3D graphics applications and applets. All of the MetApps prototype stand-alone applications use version 1.1.x or 1.2.1 of Java 3D™. For Windows and Solaris (SPARC) one can download a 1.2.1 version from Sun Microsystems. Java 3D™ on Solaris requires OpenGL while on Microsoft Windows systems Java 3D™ requires OpenGL or DirectX. OpenGL is a software interface to graphics hardware (Woo et al., 1999:2) used by many different operating systems while DirectX provides a standard graphics development platform for Microsoft Windows-based computers (Microsoft, 2001). OpenGL is standard on most computer systems, but if not, can be obtained through the OpenGL Internet site (SGI, 2001). DirectX software for Windows systems is available through Microsoft.

Unidata has tested the MetApps stand-alone prototypes mainly on the Microsoft Windows, Sun Solaris, and Linux platforms. As part of the background research for this thesis, a number of the MetApps stand-alone applications were run using Solaris version 2.8 (sometimes referred to as Solaris 8) and Microsoft Windows versions Millennium, 2000, and XP. Unidata, as far as hardware, recommends that in addition to an operating system that supports Java™ and Java 3D™, the system have a minimum of 128 MB of random access memory (RAM) available. Table 3 shows a summary of the experimental hardware environment.

Preliminary experimentation with MetApps and VisAD (which forms a critical part of MetApps) indicates that this recommendation is unrealistic. Some of the meteorological model datasets involved with this research can be quite large and will quickly overwhelm most Windows systems. Unidata also states that a video card that

Table 3. Experimental Hardware Environment

Hardware Environment		
	Windows PC	UNIX Workstation
Operating System	Microsoft Windows XP, Home Edition (Build 2600)	Sun Microsystems, Solaris 2.8
Processor	Pentium III-mobile, 1 GHz	Sparc Ultra 10, 440 MHz
Memory	512 MB	1 GB
Video Hardware	Supports DirectX hardware acceleration	Supports OpenGL hardware acceleration

supports hardware acceleration under OpenGL is recommended but not necessary for 3D applications. Preliminary research for this thesis would indicate that indeed this recommendation is also not realistic. Running the MetApps Gridded Data Viewer application on a hardware platform with a non-OpenGL video card made the 3D features unusable. Lastly, Unidata recommends processor speeds of 200Mhz on either a Sun Sparc or Intel platform as a minimum. (UCAR, 2000)

3.6 Data Requirements and Handling

In order to test the two experimental approaches to building an application described above, real meteorological model data is required. The initial approach in this research was to use MM5 model data provided by the Air Force Weather Agency. This dataset is virtually identical to the one used by the Air Force Weather Agency Technology Exploitation Branch to produce the model products available through IGrADS. The format of this data is Grid in Binary (GRIB) and can be read by GrADS

using a *ctl* file. GRIB is the gridded data standard used by the World Meteorological Organization. NCEP (the National Centers for Environmental Prediction) uses GRIB for all the files produced by their analyses. MetApps was not designed, however, to read GRIB data. Instead, netCDF is the favored data format. Thus, several attempts were made during the early course of this thesis to convert the available MM5 meteorological model data from GRIB to netCDF. These attempts proved less than satisfactory. In order to convert GRIB data to netCDF format, a UNIX-based system was used. The reason for this is that data conversion software that will run on Microsoft Windows-based personal computers is extremely limited, but available. Most standard meteorological data conversion software was designed to run exclusively on UNIX platforms. In addition, the software configuration proved quite large and complex. This approach to handling the model data was later abandoned for several reasons. The fact that a UNIX platform was going to be used to convert data made this endeavor not platform-independent and therefore did not fit in with the goals of this thesis. Second, most end-users do not have the facilities at their disposal to perform the data conversions required to view a particular model data format. Most end-users use Windows PCs and thus data conversion operations are not practical due to the time and system requirements. It was therefore determined that the most practical way to provide the necessary meteorological model data files to end-users is to convert the GRIB data into the proper format on a centrally located workstation and then serve the properly formatted data to users via an Abstract Data Distribution Environment (ADDE) server. This “proper” format is network Common Data Form (netCDF).

What exactly is netCDF? Unidata's netCDF is an interface for array-oriented data access and a library that provides an implementation of the interface. The netCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data. The freely available software implements the data model in several computer-programming languages (Rew et al., 1997). NetCDF is used in thirty countries and several groups have adopted netCDF as a standard way to represent some forms of scientific data. The netCDF-2 Java™ Library is a Java™ interface to netCDF data files and is built on the *MultiArray* package. Also included in this library is a netCDF interface to files that are accessed through a Distributed Oceanographic Data System (DODS) server. The library is composed of three packages, *ucar.nc2*, *ucar.nc2.dods*, and *ucar.ma2*. The implementation uses some of the code from the earlier NetCDF Java™ library, but the API is distinct and logically unconnected to the original. The netCDF library is completely incorporated into the MetApps library (*metapps.jar*).

The two experimental applications developed for this thesis were tested on netCDF data files obtained from Unidata and data files stored locally at the Air Force Institute of Technology. These included GRIB (in netCDF format), surface, and upper-air data. The GRIB files consisted of model output grids of various commonly used meteorological models, but not MM5.

There is no conversion of MM5 model output to netCDF format involved in the research for this thesis. The logistical and software configuration questions involved in conversion of the Air Force Weather Agency's MM5 model output into netCDF format falls outside the scope of this thesis. Also, no attempt was made to deal with database

requirements. Database issues will need to be addressed at a later time should there be a desire to adopt the use of MetApps in the operational military aviation weather community. Work has, however, been done elsewhere to address the database issues involved with the development of pure Java component-based meteorological applications. The 4DWX project has attempted to tackle the JavaTM database challenge. This effort met with some success (NCAR, 2000).

4. Results

4.1 Overview

The results obtained from construction of the two experimental applications described in Chapter 3 are discussed in this chapter. The two experimental applications are compared against the criteria listed in Chapter 3 and summarized here: The first of these criteria is that the software solution developed using the existing components be platform-independent. Second, the software solution must not be restricted to obtaining data from a single source, as has been the case in the past. Third, the solution must be robust. “Normal”-sized model data files should be able to be run on a component-based application without putting undue strain on a mid-level system. Fifth, the software solution should be user-friendly. Last, the selected software solution must be an improvement over current visualization applications used in the operational aviation weather environment. The results ended up being both positive and negative. The type of component-based approach used separates the description of the results below. Some of the results inevitably overlapped but some, such as those related to deprecation issues are applicable only to the second experimental approach that used the MetApps component library.

4.2 Application Design Approach 1

In this first experimental application design, an application was constructed using the Borland JBuilder 5 development environment. This application was designed to act as a launch pad for the three MetApps prototype stand-alone applications currently

included as part of MetApps. In other words, the compiled bytecode in the *GDV.jar*, *ImageViewer.jar*, and *InteractiveSounding.jar* took precedence over the classes contained in the MetApps component library at the time of application compilation. Figure 10 shows is a screenshot of the experimental application constructed using the first experimental approach.

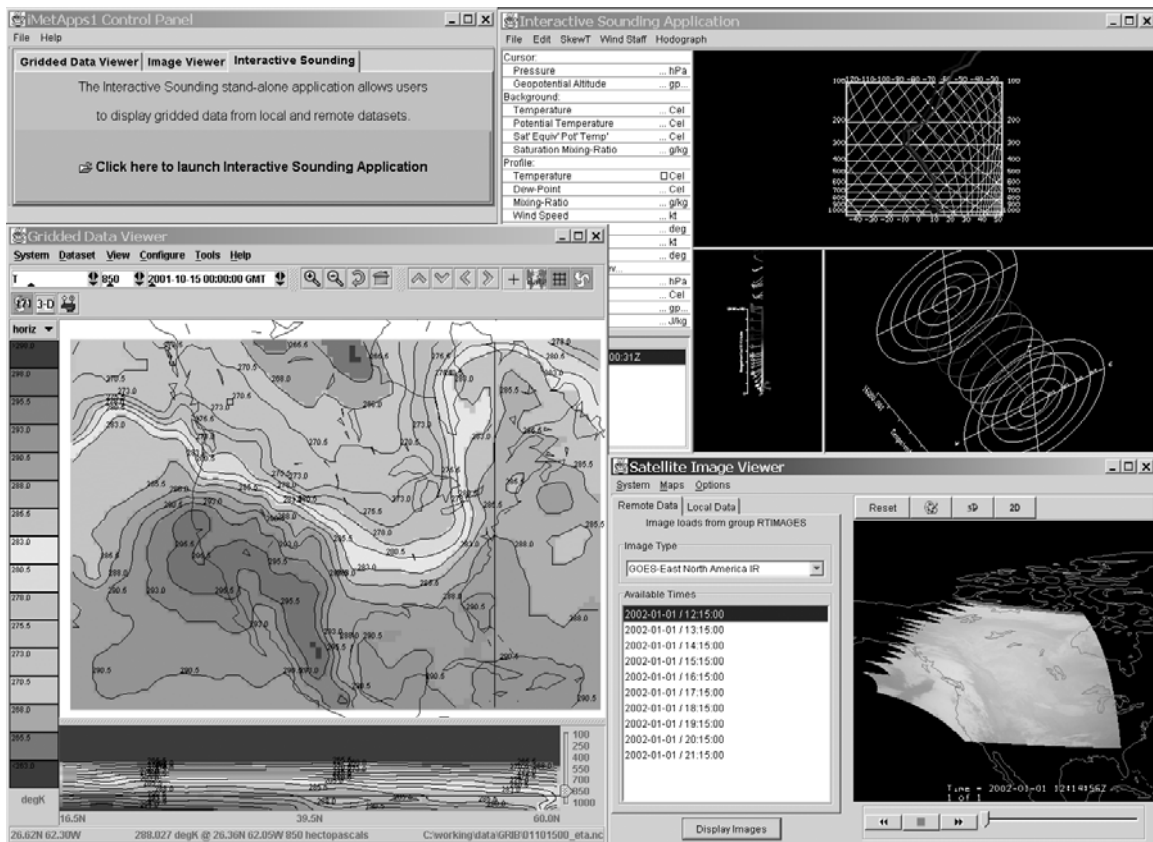


Figure 10. Experimental Application Screenshot. This screenshot shows the experimental control panel (iMetApps1) with the three MetApps stand-alone prototype applications it is intended to launch.

The results achieved by taking the first experimental approach to building an application using the MetApps components were mixed. A positive result of the

experimental approach is that the application created is completely platform-independent, as expected since pure JavaTM was used. On the whole, the behavior of the application running on the Windows PC is virtually identical to the application running on a UNIX workstation. Thus, the requirement of platform independence appears to be completely met using MetApps. A second requirement, which initially appears to be met, is that the software solution not be restricted to obtaining data from a single source. The Gridded Data Viewer (GDV) and the Interactive Sounding Application behave reliably. Both were able to successfully load local and remote Abstract Data Distribution Environment (ADDE) server data sets and allowed unrestricted interaction with the data. On the other hand there were significant problems with the Image Viewer. According to Unidata the Image Viewer can be used to read both remote and local imagery. During the experimentation phase of this thesis research, the image viewer was used to depict both United States GOES and European Meteosat satellite imagery. When loading remote imagery via an ADDE server, no problems were encountered. However, when loading local imagery, invariably problems were encountered. Within seconds of attempting to load a McIDAS AREA satellite image file the Image Viewer application would generate a JavaTM out-of-memory exception. Using a memory monitor it was determined that the application would effectively use over 300 MB of memory upon each attempt to load a local AREA file. Identical results were achieved with the Image Viewer when it was not part of the experimental indication. This, of course, effectively renders the Image Viewer useless as a tool for viewing imagery locally. However, the viewer still meets the multiple data source requirement.

The third requirement was that the solution must be robust and that normal-sized model data files should be able to be run on a component-based application without putting undue strain on a mid-level Windows PC or an average UNIX workstation. This requirement was not met. Running any of the separate MetApps prototype stand-alone applications by themselves already pushed the hardware limits of the experimental environment. When two or more applications ran simultaneously, resources became extremely strained and caused memory errors even on systems with 512 and 1024 MB of RAM. Another problem with the stand-alone prototypes is that they have a tendency to throw uncaught VisAD and JavaTM remote method invocation exceptions. A JavaTM exception is an abnormal condition that arises in a code sequence at runtime. The exceptions that are thrown appear to be part of the normal functioning of the MetApps prototypes, but they can prove awkward when developing an application and should be dealt with appropriately. The exceptions were dealt with by some exception handling code in the experimental applications that were constructed for this thesis. The presence of these undocumented exceptions brings attention to how experimental MetApps still is.

The fourth requirement was that the solution must be extensible or upgradeable. An application built on the MetApps library should be relatively easy to adapt. Through experimentation this requirement was met. An awkward side effect initially with the first experimental application was that every time a subcomponent was closed the entire application would terminate. This appeared to be a result of the way the MetApps prototype stand-alone prototypes were coded. To demonstrate how simple it was to adapt the code to deal with this problem, a relatively crude form of component engineering was applied to the problem. An updated component (or actually a JavaTM class) was

constructed to replace the component causing the problem. The *interSounding.java* class replaced the *SoundingApplication.java* class in the Interactive Sounding Application, which was causing the undesirable behavior. This successful alteration of the code shows how extensible and upgradeable MetApps actually can be. Hence, the upgradeability requirement is met, but some recoding may be required if changes are made to the application.

The fifth requirement, dealing with user-friendliness, was met by the construction of a graphical user interface (GUI) using the Borland JBuilder environment. This task was rather straightforward and took advantage of the swing widgets that come standard with the Java™ 2 SDK. The end result was a utilitarian interface that is simple, functional, and eliminated the need for a command line.

The final requirement was that the MetApps solution must be an improvement over current visualization applications used in the operational aviation weather environment. Regrettably this requirement could not be met. Although the experimental MetApps-based application offers some improved functionality as far as interaction with data sets it cannot reproduce the products that are currently being generated using GrADS. One of the most popular aviation weather products is the meteorogram. There are plans to incorporate functionality in MetApps that will support meteorogram generation and time-series in general; however, the current version of MetApps does not support these capabilities. Hence, MetApps does not meet this requirement. In addition, the experimental applications constructed for this thesis were very large. Construction of Java™ application archives resulted in a 28.1 MB Java™ archive (JAR) file for the first approach and 16.6 MB size for the second experimental approach. These JARs only

included the so-called required classes. If there were an attempt to turn these applications into web-applications or applets, the result would be a monstrosity that would take up an inordinate amount of bandwidth. This largeness is not a configuration problem but rather the result of the large number of libraries that MetApps requires to function properly. To currently obtain products via JAAWIN all one needs is a JavaTM-enabled browser and a modem.

A particularly disturbing result of attempting the incorporation of the MetApps prototypes in an experimental application was related to the Gridded Data Viewer (GDV). There appear to be a number of conflicts between the GDV and the Swing widgets used to construct the experimental application user interfaces. These conflicts persisted throughout six months of successive iterations of the experimental applications. Finally, a decision was made to run the GDV in a completely separate process, bypassing the conflict entirely. Hence, the GDV was no longer part of a single application.

4.3 Application Design Approach 2

The second application design approach yielded similar results to those found with the first. In this approach however the experimental MetApps application was built from source code rather than the prototype applications. This added some additional interesting angles to the results.

The results achieved by taking the second experimental approach to building an application using the MetApps library were mixed. As with the first approach, the application created is completely platform-independent. Behavior of the application running on the Windows PC is identical to the one running on a UNIX workstation. The

requirement of platform independence appears to be completely met. The second requirement is that the software solution not be restricted to obtaining data from a single source. The Interactive Sounding Application behaved reliably as with the first approach. On the other hand there were significant problems with the Image Viewer. Two of the four classes required to construct the Image Viewer Application were completely omitted from the MetApps source code libraries. Hence, the capabilities of the Image Viewer could not be replicated without inclusion of the Image Viewer prototype stand-alone application JAR files. This however would have made the second experimental approach identical to the first, making any attempt at a second, different, approach irrelevant. It can be concluded therefore that the image viewer can actually not be generated from the MetApps source code. Curiously, there exists no documentation as to why these classes were omitted from the source code. Such an omission can be attributed to poor documentation or software engineering practices (configuration management in particular).

The third requirement was that the solution be robust and that normal-sized model data files should be able to be run on a component-based application without putting undue strain on a mid-level system. This requirement was not met and results were almost identical to those experienced with the first approach. As with the first experimental approach the components throw uncaught VisAD and JavaTM remote method invocation exceptions. The exceptions were dealt with in the same way as with the first approach, by exception handling code. The presence of undocumented exceptions in the source code is further indication of the experimental nature of MetApps.

The fourth and fifth requirements dealing with extensibility/upgradeability and user-friendliness appear to be mostly met. The ability to customize an application using this approach is much greater than with the first. A developer is not restricted here by what is contained in the JAR of a particular MetApps application. The developer can pick and choose the classes from the component library necessary to meet a particular requirement. The application constructed using this second approach in this thesis is very similar in its capabilities to the application constructed using the first. The sixth requirement is not satisfied since the application constructed did not appear to be a significant improvement over current applications. Due to time constraints and the poor state of the MetApps documentation, extensive customization of an application was not attempted as part of this research. Whether or not an application can be built that is an improvement over current visualization applications is dependent on a number of factors; among these are the skill and patience of the individual software developer.

The most troubling result of taking this second approach to building an experimental MetApps application is related to deprecation. Deprecated methods all produce warnings that let a developer know that he or she is advised to use the newer method. Deprecation allows an industry to gradually remove parts of software that are intended to be phased-out but which cannot be phased-out immediately because too many programs depend on the retired parts. The deprecation problem stems from the fact that the prototype applications and components in MetApps were developed sequentially over a period of several years. As one part of MetApps was completed it was added to the MetApps library and development was begun on the next application or component. Deprecation is particularly bad with the VisAD packages upon which MetApps heavily

depends. The VisAD project and library has been under continuous development completely apart and separate from MetApps and is still undergoing changes. The result of this parallel development has been that some classes upon which MetApps depends no longer exist in the most recent version of the VisAD library. For instance, the Interactive Sounding Application was completed around July of 2000. If one were to compile and run this application and then attempt to load an upper-air sounding, one would not be able to render a 3D hodograph. The reason for this is that the required classes to perform this action have been removed from VisAD. In fact, a total of five classes, two interfaces, two exceptions, twenty-six constructors, and well over fifty methods have been deprecated in VisAD since the project's inception. This deprecation problem does not manifest itself in the first experimental approach of this thesis. The reason for this is that the version of VisAD that was current at the time the Interactive Sounding Application was completed, is wholly included in the application JAR.

On a more positive note, the size of the experimental application resulting from building from source code is smaller than that in the first approach used. However, at 16.6 MB even this improvement in the size of the application does not make it very Internet-friendly.

4.4 Summary of Results

A summary of the experimental results is provided in Table 3. The question of whether or not the requirements were met is answered with YES, NO, or MAYBE.

The JavaTM source code for the experimental applications constructed as part of this thesis is included in Appendix C.

Table 4. Summary of Experimental Results

Requirement	Design Approach 1	Design Approach 2
1. Platform Independent	YES	YES
2. Multiple Data Sources	YES	YES
3. Normal Behavior	NO	NO
4. Upgradeable	YES	YES
5. User-Friendly	YES	YES
6. Improvement over Current Capabilities	NO	MAYBE

5. Conclusions and Recommendations

Work centers today that provide meteorological imagery products to military users depend on state-of-the-art computing systems. These state-of-the-art systems are actually large monolithic, platform-dependent applications. McIDAS, GrADS, and Vis5D fall into this category even though there are ways to port them to other platforms. Some progress has been made in factoring out common code in these systems into common libraries or APIs (Application Programmer's Interfaces) such as Vis5D. With each successive software release, a little more of the code is factored-out into the shared libraries. However, because of other issues such as backward compatibility and legacy systems support, these attempts at modularization have been mostly thwarted. The end result of this process is some very large (one gigabyte sized) systems such as McIDAS. Scientists using Fortran did the pioneering work in these meteorological systems. Their goal was to implement mathematical algorithms as efficiently as possible. The lack of proper software-engineering practices in the development of these systems is evident. This limits the flexibility and reusability of these systems.

Since the mid 90's there has been a proliferation of stand-alone workstations and Microsoft Windows PCs, which in many cases have better graphics capabilities than large dedicated meteorological visualization systems. A few "hero" meteorologist-programmers took on the challenge of porting parts of the large meteorological systems to the new stand-alone hardware and Internet community. These "heroes" in many cases have a much better grasp of good software development practices than the original creators of the systems they are tackling. An example of such an adaptation can be seen

at the Air Force Weather Agency Technology Exploitation Branch. They have achieved the successful combination of GrADS with a JavaTM interface to provide user-customized aviation weather products to customers on-demand. Even these valiant efforts, though, will not prevent the eventual demise of systems such as GrADS. Eventually, dependence on third parties, legal issues with the same, and the complexity and size of the systems makes maintenance and the addition of new features extremely difficult.

Into this world of monolithic platform-dependent systems entered the concept of component-based software engineering (CBSE). Unidata extended the netCDF model to JavaTM and projects such as VisAD implemented the concepts found in CBSE. These were followed-up by projects such as MetApps and 4DWX VMET that have exploited the components found in these libraries and in turn built more complex components. Although brave and innovative, these forays into the cutting edge of geophysical visualization application development are still largely experimental.

There are important factors that must be considered before adopting the use of software components and a component-based software solution. Although the quality improvement of products built using CBSE is well documented (Pressman, 2001) there are drawbacks to the approach. The development of component-based software is harder than traditional software development. In addition, there are the up-front costs involved in retraining personnel in the new software development approach. Of course, in the case of the Air Force Weather Agency Technology Exploitation Branch, development of new components would be minimal. Instead, emphasis would be placed on re-use of existing components in an effort to minimize costs related to software development. The problem that would be encountered here is that in order to use software components effectively, a

developer must have extensive knowledge of the component library in question. This problem was encountered over the course of this thesis research. Much time was spent simply analyzing the class structure of the MetApps and associated libraries rather than constructing the actual experimental applications. Component-based development using poorly designed or documented software components, such as those in MetApps, is therefore filled with pitfalls. With the current state of component-based development technologies and the limitations of components, component-based development by personnel with limited computing expertise should only be attempted with great caution.

The prototype applications built with MetApps show that the experimental designs used in this research will work. From the progress that has been made thus far, it can be seen that some day JavaTM applications built from components such as those found in the MetApps library may be part of the ideal operational weather system. That day, regrettably, is not today. Although prototype applications based on a component library, in this case MetApps, were successfully constructed over the course of this thesis research, they are by no means of operational caliber. As can be seen from the results discussed in Chapter 4, Unidata MetApps is still too experimental to be used in an operational environment. It is yet too unpredictable and unstable, and lacks the robustness required of an operational system. Systems such as GrADS, even though not ideal, have the stability and predictability that are required of operational systems. The current iteration of MetApps is best suited for the academic research environment. One could, however, reasonably conclude that at the pace at which progress is being made with the development of MetApps, a more stable and non-experimental version should become available within the next two years. Although it is recommended that close

attention should be paid as to further developments in MetApps, MetApps should not be included in any operational aviation weather products generation software system at this time.

Over the course of this research, JavaTM component libraries other than MetApps were encountered. One that shows great promise is the Visualization for Algorithm Development (VisAD) library. The research community has successfully used VisAD. Future research might include the development and testing of customized military aviation product generation software solutions built entirely from VisAD JavaTM components. In addition, future research should look more closely into possible ways of dealing with the numerous database issues involved in using JavaTM to manipulate and display large meteorological model data sets.

Appendix A: Abbreviations

2D	Two-dimensional
3D	Three-dimensional
4DWX	Four-dimensional Weather System
5D	Five-dimensional
ADDE	Abstract Data Distribution Environment
AFWA	Air Force Weather Agency
AFWIN	Air Force Weather Information Network
API	Application Programming Interface
ATEC	Army Test and Evaluation Command
CBSE	Component-Based Software Engineering
COLA	Center for Ocean-Land-Atmosphere Studies
COTS	Commercial Off-the-shelf Software
DAO	Data Assimilation Office
DNXT	AFWA Technology Exploitation Branch
DODS	Distributed Oceanographic Data System
GB	Gigabyte
GDV	Gridded Data Viewer
GHz	Gigahertz
GOES	Geosynchronous Operational Environmental Satellite
GrADS	Grid Analysis and Display System
GRIB	GRid in Binary

ICAO	International Civil Aviation Organization
IMaST	Interactive Meteorogram and Skew-T
JAAWIN	Joint Air Force and Army Weather Information Network
JAR	Java™ Archive
km	kilometer
MB	Megabyte
MHz	Megahertz
MetApps	Meteorological Applications
MGWDS	Mesoscale Gridded Window Display System
MM5	Mesoscale Model 5
NCAR	National Center for Atmospheric Research
netCDF	network Common Data Form
PC	Personal Computer
RWM	Relocatable Window Model
SGI	Silicon Graphics Incorporated
SSEC	Space Science and Engineering Center
UCAR	University Corporation for Atmospheric Research
UMADA	Unidata MetApps Discussion Area
VisAD	Visualization for Algorithm Development
VMET	Visual Meteorology Tool

Appendix B: Unidata MetApps Detailed Analysis

B.1 Overview

Provided here is an in-depth analysis of MetApps, which forms the basis for the experiments that are performed as part of this thesis. MetApps (Meteorological Applications) is available both as a JavaTM library and as a number of stand-alone applications. Without fully elaborating on what these two separate (but related) versions of MetApps are composed of, it would not be possible to fully explain the design approach taken in this thesis. Each of the MetApps “versions,” both the library and applications, has its merits and drawbacks when used in building a meteorological application and these will be discussed further in sections of this document dealing with experiment design. What follows in the next two sections is a breakdown of the MetApps library and the available MetApps stand-alone applications.

B.2 Design of MetApps Prototype Stand-alone Applications

An application is a piece of software that is intended to be used in a stand-alone manner. Since the inception of the Unidata MetApps project the number, type, and even the names of stand-alone MetApps applications has remained fluid. Among the earlier stand-alone applications (during 1999) was the Surface Observation Viewer (SOV) Application. Although most of the JavaTM classes associated with the SOV are still present some classes critical to the proper operation of this application have been deprecated. In addition, there are applications currently being developed through Unidata such as the Radar Data Viewer, which will not be complete upon completion of this

thesis. Upon the writing of this thesis, Unidata has available three MetApps stand-alone applications: Interactive Sounding, Image Viewer, and Gridded Data Viewer.

The following libraries are being used in the MetApps stand-alone prototype applications: JavaHelp™, VisAD, MetApps, and Ancillary Data Files. The JavaHelp™ software library is a full-featured, platform-independent, extensible help system that enables developers and authors to incorporate online help in applets, components, applications, operating systems, and devices. It is being used as the interface to the on-line documentation for the prototypes. VisAD is a Java™ class library for interactive and collaborative visualization and analysis of numerical data. Its data model and display capabilities are being used in several of the prototypes. The MetApps prototypes also use a standard library of classes developed by Unidata called the MetApps library. In addition to the libraries above, some ancillary data files (icons, enhancements, map files) are used by the applications and are packed together in the Ancillary Data Files library. The libraries are available from Unidata as four JAR (Java™ Archive) files named *jh.jar*, *visad.jar*, *metapps.jar*, and *auxdata.jar*. All the libraries contain primarily Java™ bytecode except for *auxdata.jar*, which contains a variety of other formats. It should be pointed out that in addition to these libraries, each specific MetApps stand-alone application, available as a download from Unidata, may require its own additional libraries (or JARs).

B.2.1 Interactive Sounding Application

A MetApps stand-alone application that shows much promise for incorporation in an aviation meteorological software suite is the MetApps Interactive Sounding

Application. The Interactive Sounding Application displays RAOB (radiosonde observation)-like sounding data in aerodynamic diagrams (e.g. Skew T chart, wind hodograph). Unidata specifies that this particular application is not intended to become a finished, stand-alone application (though that is not precluded). Rather, its purpose was to present several software components in a convenient framework for the purpose of refining the components (rather than the application). A component is a piece of software that is not intended to be used in a stand-alone manner, but, rather, is designed to interact with other components. Several interacting components may form an application and a component may be used in more than one stand-alone application. (UMADA, 2001)

One of the Sounding Application components, the wind profile component, comprises two sub-components: a 3D hodograph and a wind staff. In the hodograph, the mean-wind is displayed as a point. In the wind staff, the mean-wind is displayed as a white wind-arrow. The Mean-wind computation is only possible if there is no wind data outside the domain of the thermodynamic data. A second component, the sounding selector, is invoked by selecting the "Open sounding..." item from the *file* menu. The button labeled *Select netCDF Upper Air File* is for selecting upper air data in a netCDF file that is available on a local drive. Shown in Figure 11 below is a screenshot of the MetApps Interactive Sounding Application showing the different subcomponents of this prototype stand-alone application.

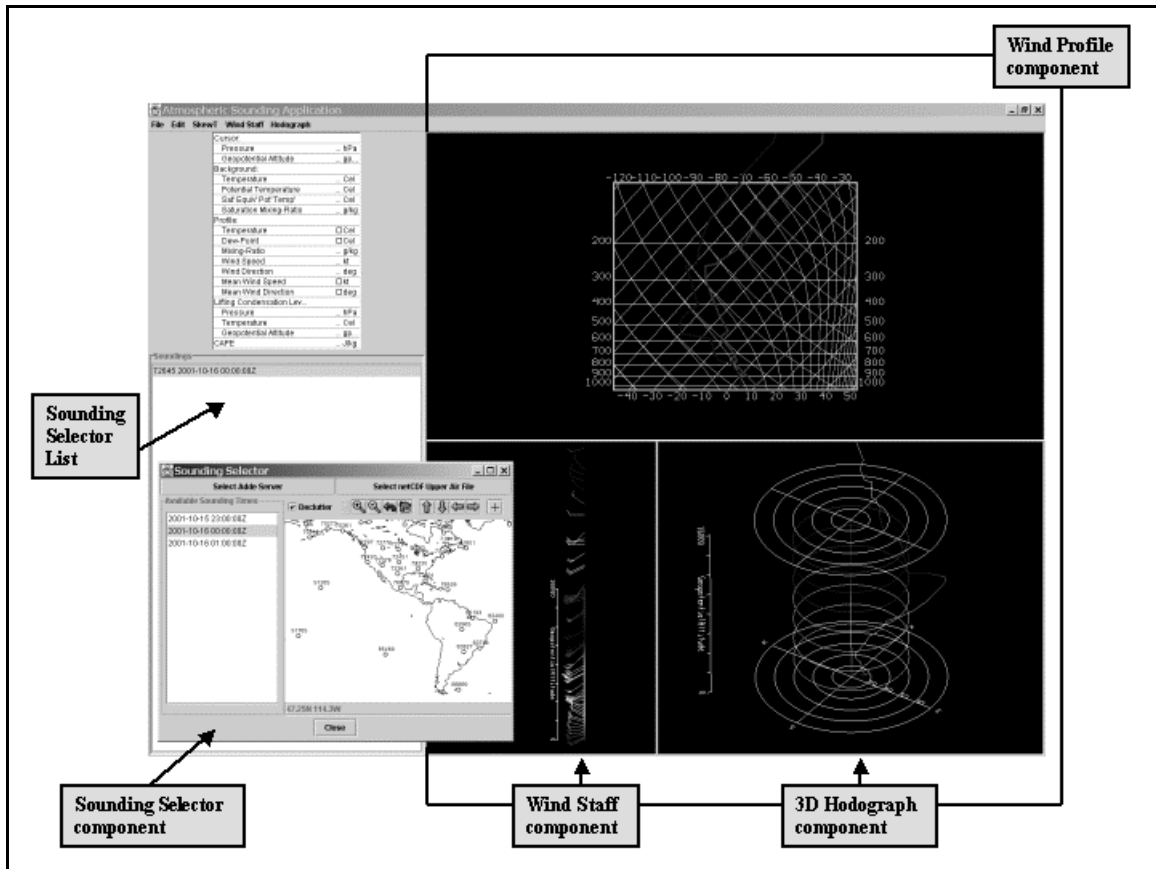


Figure 11. Interactive Sounding Screenshot. This screenshot of the MetApps prototype stand-alone Interactive Sounding Application shows its different subcomponents.

A more promising option for selecting a sounding is using the *Select ADDE Server* button to obtain data from a McIDAS ADDE (Abstract Data Distribution Environment) server. ADDE allows a workstation to act as a client, efficiently accessing data from multiple McIDAS servers. There are several advantages to using ADDE to access image data. First, there is no need to have the data reside on the local machine. No dedicated connections are needed and access can be accomplished over a network connection or modem. Data can be accessed from within a military base for example, or from an ADDE server on the other side of the world. Second, ADDE servers can be used

to access data in formats other than McIDAS AREA file format. Currently, Unidata supports servers for NIDS, NOWRad, and GINI format in addition to McIDAS AREA format. Third, the user is provided data redundancy and possibly access to data sets that are not available locally.

Finally, the Sounding Selection List component appears in the lower-left corner of the main display of the Sounding Application. It lists the soundings that have been added to the Interactive Sounding system via the Sounding Selector component and allows selection of soundings for display, computation, or removal. The Interactive Sounding Application is completely self-contained and requires no exterior component libraries other than those already mentioned previously.

B.2.2 Gridded Data Viewer (GDV) Application

The second available MetApps stand-alone application is the Gridded Data Viewer, or GDV. The GDV is a general tool for displaying geogrids. In a geogrid, individual grid elements are called grid cells. A geogrid has a data value for each grid cell, unless the data is missing. Missing data is mapped to the background color of the display. Color mapping is a simple display method where data intervals are assigned to colors. Color mapping allows the user to visualize geogrids in a way that directly reflects the data values of the geogrid. The 2001 version of the GDV is specialized to display model data output, and may or may not be extended to display satellite imagery. Currently the GDV can read netCDF files using NUWG, CSM, COARDS, and GDV netCDF conventions. (UMADA, 2001) Figure 12 shows a screenshot of the typical product the GDV is capable of producing.

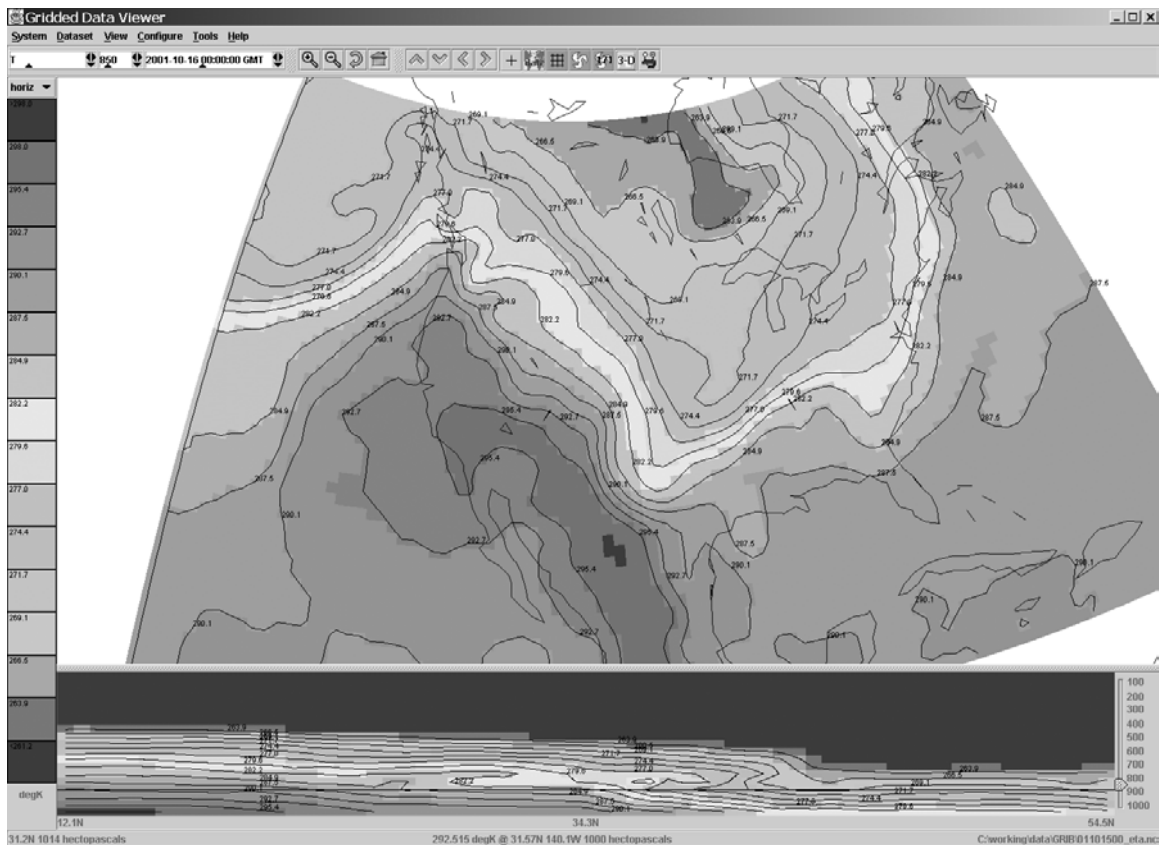


Figure 12. GDV Screenshot. This screenshot shows the gridded Eta meteorological model 850mb 24-hour temperature forecast displayed on the MetApps Gridded Data Viewer (GDV).

Current features of the GDV include the ability to display

- horizontal and vertical planes (or slices) of geogrids
- data on various projective geometries; projections are parameterized and extensible
- optional contours of the data
- data in a 3D viewer
- simple looping

The GDV has been tested on Windows, Linux, and Solaris SPARC systems and can read most forms of netCDF data, although not all (UMADA, 2001). To actually be able to use the GDV stand-alone application some additional library files are required apart from the ones previously mentioned that are required for all MetApps applications. These are the Distributed Oceanographic Data System (DODS), Frequently Asked Questions Organizer (FAQO), JavaTM Document Object Model (JDOM), and Xerces JavaTM libraries.

The DODS JavaTM library (available as *dods.jar*) provides support to the GDV for accessing data on DODS servers. DODS simplifies aspects of scientific data networking by making local data accessible to remote locations regardless of local storage format. FAQO is a tool for online technical support groups, who exchange questions, answers, and information through email, Netnews, or other computer-mediated forums. It makes it easier for support group members to search archives for previously asked (and answered) questions, and to collaboratively organize and manage those archives. FAQO uses advanced information retrieval techniques to allow natural language searches for relevant documents. The FAQO library (*faqo.jar*) needed by GDV only consists of a couple of user interface classes. The JDOM JavaTM Library contains a relatively new JavaTM API for reading, writing, and manipulating XML from within JavaTM code. JDOM brings with it the capability of providing a full document view with random access but does not require the entire document to be in memory. The JDOM API allows for future flyweight implementations that load information only when needed (Hunter, 2001). Finally, the Xerces JavaTM library (*xerces.jar*) brings to the GDV application an open-source XML

parser, which is available through the Apache XML Internet site (Apache Software Foundation, 2001).

B.2.3 Image Viewer Application

The Image Viewer prototype stand-alone application (shown in Figure 13 below) allows users to display image data from local and remote datasets. Although the Image

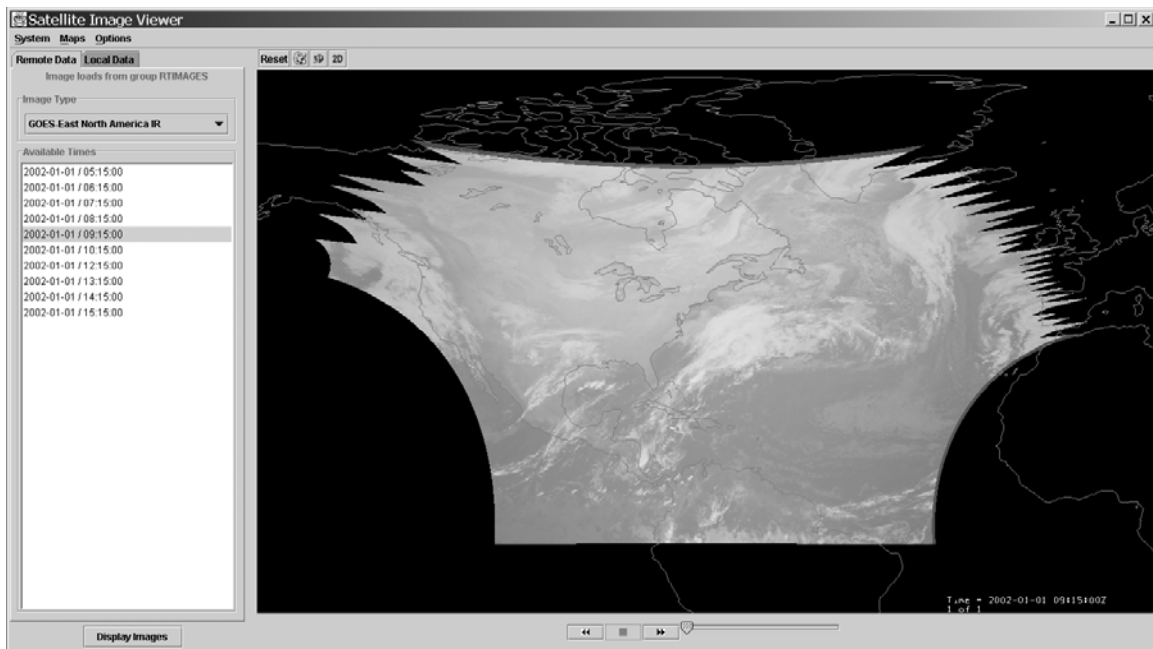


Figure 13. Image Viewer Screenshot. This screenshot of the MetApps prototype stand-alone Image Viewer Application shows the viewing window and the remote/local image selection components.

Viewer application was initially developed as a satellite data viewer, the current viewer can be used to display both radar and satellite imagery. As far as special requirements, the Image Viewer application needs only the standard libraries used for the MetApps applications installed correctly. At the present time, only McIDAS AREA files can be

opened through the local data access methods. Unidata plans to incorporate support for other image data formats in future versions of the Image Viewer application (UMADA, 2001). Some of these other formats can be accessed through ADDE servers (discussed above).

The Image Viewer stands out as a stand-alone application because of its diminutive size. Shown in Figure 14 below, the entire application consists of a three Java™ classes and some enhancements, icons, and map files. Everything else in the Image Viewer application is accomplished by importing classes from the VisAD and MetApps Java™ libraries. The end result is a display application consisting of a VisAD

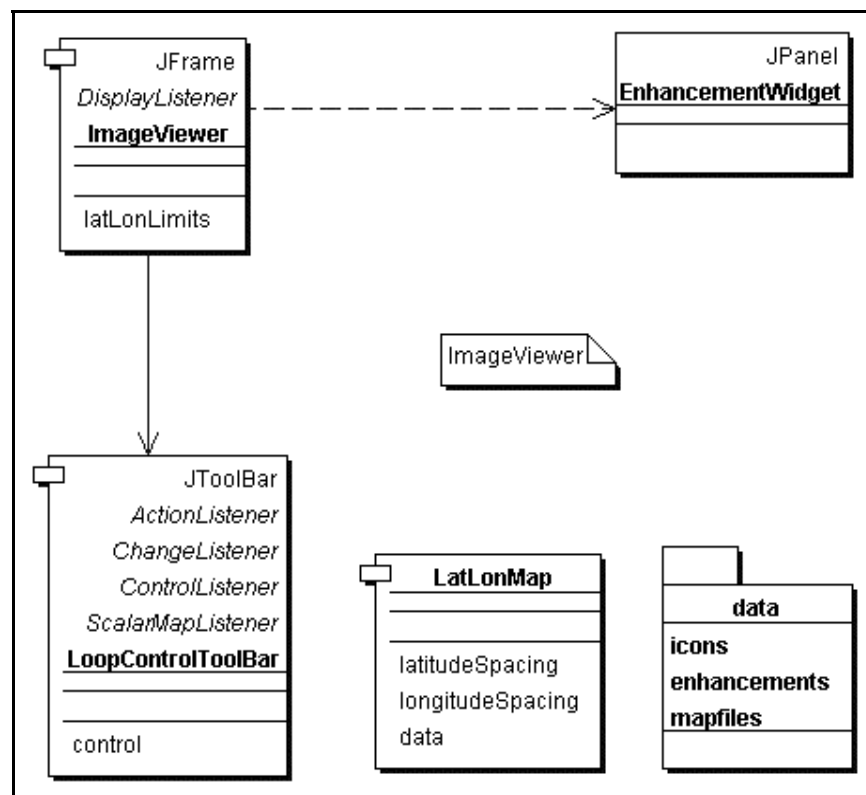


Figure 14. Image Viewer Class Diagram. This class diagram of the Image Viewer Application shows the simplicity of this particular MetApps prototype stand-alone application.

display window, which can be either two or three-dimensional, and a couple of image-selection components (or widgets). Size is a consideration when one plans to use an application over the Internet with limited available bandwidth.

B.3 Design of MetApps Library

The MetApps library consists of the base classes required to construct pure JavaTM components that are combined into applications. The library is divided into a number of major packages. These are *unidata*, *units*, *util*, *visad*, *ma2*, *multiararray*, *nc2*, and *netcdf*. Figure 15 shows how the sub-packages with the MetApps library are linked together. The dashed arrows show the dependencies between packages. For example, *unidata* depends on *util*, while *visad* and *unidata* are interdependent. Although *unidata* is presented within the MetApps library as a JavaTM package, it is not really a package at all but rather the skeleton for organizing a series of important sub-packages each of which forms a critical component of the MetApps library. It is these critical packages, like the *unidata.gridviewer* (which is the core of the GDV application) that are combined to build useful MetApps applications. The *units* package provides support for parsing and formatting string unit specification, converting numerical values between compatible units, and performing arithmetic on units (such as dividing one unit by another). *util* simply consists of an interface and two classes used in logging server activity. The reader should note that the *visad* package here is not the same as the VisAD library mentioned in Chapter 2. A developer may be tempted to replace this package with the VisAD library, but this temptation should be resisted. The *visad* package provides

support for hiding some of the complexity of the large VisAD package that is required to build MetApps applications.

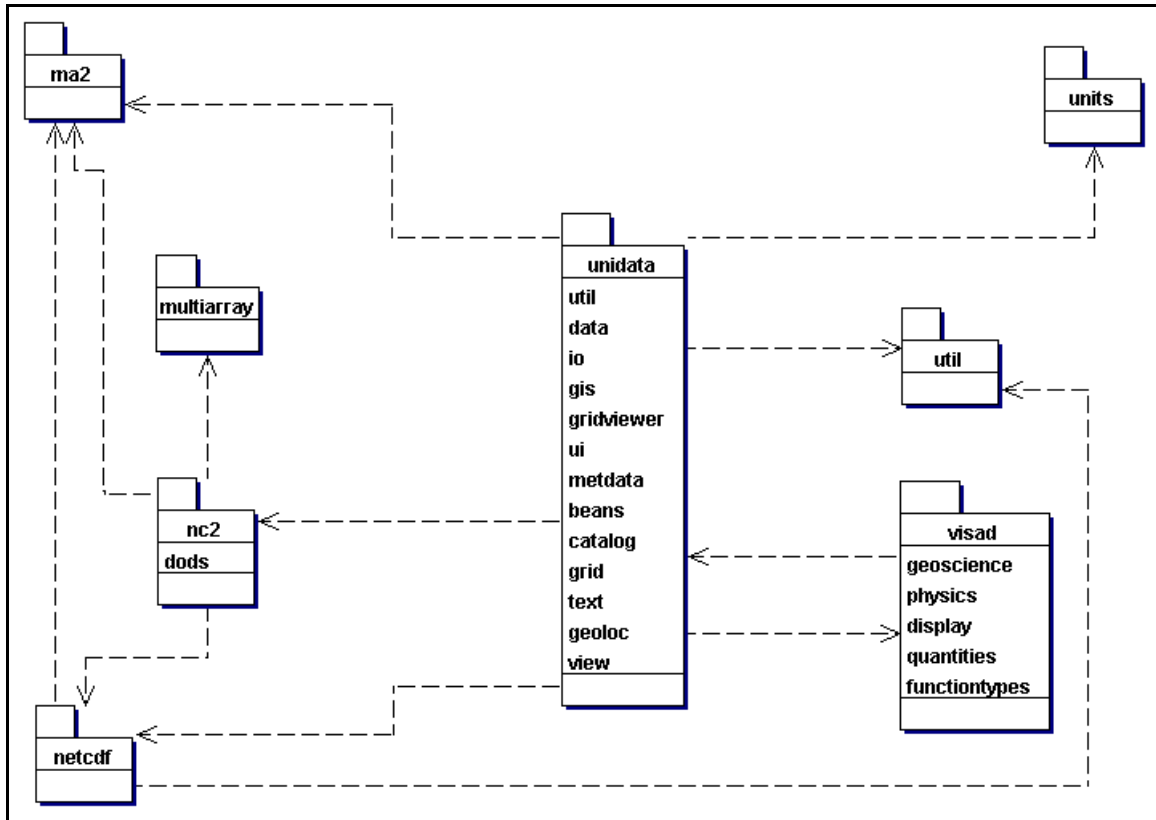


Figure 15. MetApps Library Package Diagram. This figure shows the principal *ucar* packages of the MetApps library and their interdependencies. Note the large number of sub-packages contained in *unidata*.

The last four of the packages listed above are actually the netCDF-2 library, which has been wholly and permanently incorporated into the MetApps library. The *ma2* package is actually the *MultiArray* package. The *MultiArray* package is a stand-alone Java™ package for multidimensional arrays of primitive types. *ma2* is the abstraction for multidimensional arrays of primitives with data stored in memory, on a remote or local

input/output device. The *multiararray* package provides an abstraction for multidimensional array access, some concrete implementations, and ways to view a MultiArray as if it had a different structure. The *ucar.nc2* package is an experimental API for netCDF files, and the *netcdf* package provides an abstraction for sampled functions between multidimensional spaces.

Appendix C: Application Source Code

C.1 Application1.java Source Code

```
package application1;

import javax.swing.UIManager;
import java.awt.*;

/**
 * Title:          iMetApps Experimental Application - Design 1
 * Description:    This is an experimental application using design 1. This experimental
 *                application uses Unidata MetApps prototype stand-alone applications
 *                GDV.jar, SoundingApplication.jar, and ImageViewer.jar.
 * Created:       December 2001
 * Company:      AFIT/ENP
 * @author       Captain Harmen P. Visser
 * @version      1.0
 */

public class Application1 {
    boolean packFrame = false;

    /**Construct the application*/
    public Application1() {
        Frame1 frame = new Frame1();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from their layout
        if (packFrame) {
            frame.pack();
        }
    }
}
```

```

    }
    else {
        frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
        frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);
}
/**Main method*/
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    new Application1();

}
}

```

C.2 Frame1.java Source Code

```
package application1;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import com.borland.jbcl.layout.*;
import test.*;
import ucar.unidata.view.sounding.*;
import java.rmi.RemoteException;
import visad.VisADException;
import ucar.unidata.gridviewer.*;

/**
 * Title:      iMetApps Experimental Application - Design 1
 * Description: This is an experimental application using design 1. This experimental
 *              application uses Unidata MetApps prototype stand-alone applications
 *              GDV.jar, SoundingApplication.jar, and ImageViewer.jar.
 * Created:    December 2001
 * Company:    AFIT/ENP
 * @author     Captain Harmen P. Visser
 * @version    1.0
 */

public class Frame1 extends JFrame {
    JPanel contentPane;
    JMenuBar jMenuBar1 = new JMenuBar();
    JMenu jMenuFile = new JMenu();
    JMenuItem jMenuFileExit = new JMenuItem();
    JMenu jMenuHelp = new JMenu();
    JMenuItem jMenuHelpAbout = new JMenuItem();
    JToolBar jToolBar = new JToolBar();
    ImageIcon image1;
    ImageIcon image2;
    ImageIcon image3;
    JLabel statusBar = new JLabel();
    BorderLayout borderLayout1 = new BorderLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    Border border1;
    JTextField jTextField1 = new JTextField();
    JToggleButton jToggleButton1 = new JToggleButton();
    JPanel jPanel1 = new JPanel();
}
```

```

JPanel jPanel2 = new JPanel();
JPanel jPanel3 = new JPanel();
JLabel jLabel2 = new JLabel();
JLabel jLabel1 = new JLabel();
JButton jButton2 = new JButton();
PanelLayout panelLayout1 = new PanelLayout();
TitledBorder titledBorder1;
TitledBorder titledBorder2;
JLabel jLabel3 = new JLabel();
JLabel jLabel4 = new JLabel();
JButton jButton3 = new JButton();
JButton jButton4 = new JButton();
JLabel jLabel5 = new JLabel();
JLabel jLabel6 = new JLabel();
PanelLayout panelLayout2 = new PanelLayout();
PanelLayout panelLayout3 = new PanelLayout();

/**Construct the frame*/
public Frame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**Component initialization*/
private void jbInit() throws Exception {
    image1 = new ImageIcon(application1.Frame1.class.getResource("openFile.gif"));
    image2 = new ImageIcon(application1.Frame1.class.getResource("closeFile.gif"));
    image3 = new ImageIcon(application1.Frame1.class.getResource("help.gif"));

    //setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.getResource("[Yo
ur Icon]"))));
    contentPane = (JPanel) this.getContentPane();
    border1 = BorderFactory.createLineBorder(Color.gray,2);
    titledBorder1 = new TitledBorder("");
    titledBorder2 = new TitledBorder("");
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(460, 263));
    this.setTitle("iMetApps1 Control Panel");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");

```

```

jMenuFileExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
    }
});
jMenuHelp.setText("Help");
jMenuHelpAbout.setText("About");
jMenuHelpAbout.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
    }
});
jTabbedPane1.setBackground(SystemColor.inactiveCaptionText);
jTabbedPane1.setFont(new java.awt.Font("Dialog", 1, 14));
jTabbedPane1.setBorder(border1);
jTabbedPane1.setMinimumSize(new Dimension(480, 150));
jTabbedPane1.setPreferredSize(new Dimension(500, 240));
jTabbedPane1.addFocusListener(new java.awt.event.FocusAdapter() {

});
jTextField1.setText("jTextField1");
jToggleButton1.setText("jToggleButton1");
jPanel1.setBackground(Color.lightGray);
jPanel1.setMinimumSize(new Dimension(470, 155));
jPanel1.setPreferredSize(new Dimension(470, 155));
jPanel1.setLayout(panelLayout3);
jPanel2.setBackground(Color.lightGray);
jPanel2.setLayout(panelLayout2);
contentPane.setPreferredSize(new Dimension(500, 240));
jPanel3.setBackground(Color.lightGray);
jPanel3.setLayout(panelLayout1);
jLabel2.setFont(new java.awt.Font("Dialog", 0, 16));
jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
jLabel2.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel2.setText("to display image data from local and remote datasets.");
jLabel1.setFont(new java.awt.Font("Dialog", 0, 16));
jLabel1.setAlignmentX((float) 0.5);
jLabel1.setAlignmentY((float) 1.0);
jLabel1.setMaximumSize(new Dimension(300, 60));
jLabel1.setMinimumSize(new Dimension(300, 60));
jLabel1.setPreferredSize(new Dimension(470, 30));
jLabel1.setToolTipText("");
jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
jLabel1.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel1.setText("The Image Viewer stand-alone application allows users");

```



```

jButton2.setBackground(Color.gray);
jButton2.setFont(new java.awt.Font("Dialog", 1, 16));
jButton2.setBorder(BorderFactory.createRaisedBevelBorder());
jButton2.setMaximumSize(new Dimension(419, 27));
jButton2.setMinimumSize(new Dimension(419, 27));
jButton2.setPreferredSize(new Dimension(419, 27));
jButton2.setIcon(image1);
jButton2.setText("Click here to launch Image Viewer Application");
jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        jButton2_mouseClicked(e);
    }
});
jLabel3.setText("The Gridded Data Viewer stand-alone application allows users");
jLabel3.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
jLabel3.setToolTipText("");
jLabel3.setPreferredSize(new Dimension(470, 30));
jLabel3.setMinimumSize(new Dimension(300, 60));
jLabel3.setMaximumSize(new Dimension(300, 60));
jLabel3.setAlignmentY((float) 1.0);
jLabel3.setAlignmentX((float) 0.5);
jLabel3.setFont(new java.awt.Font("Dialog", 0, 16));
jLabel4.setText("to display gridded data from local and remote datasets.");
jLabel4.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
jLabel4.setFont(new java.awt.Font("Dialog", 0, 16));
jButton3.setText("Click here to launch Gridded Data Viewer Application");
jButton3.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        jButton3_mouseClicked(e);
    }
});
jButton3.setBorder(BorderFactory.createRaisedBevelBorder());
jButton3.setIcon(new ImageIcon(Frame1.class.getResource("openFile.gif")));
jButton3.setFont(new java.awt.Font("Dialog", 1, 16));
jButton3.setBackground(Color.gray);
jButton4.setBackground(Color.gray);
jButton4.setFont(new java.awt.Font("Dialog", 1, 16));
jButton4.setBorder(BorderFactory.createRaisedBevelBorder());
jButton4.setIcon(new ImageIcon(Frame1.class.getResource("openFile.gif")));
jButton4.setText("Click here to launch Interactive Sounding Application");
jButton4.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e)
    {

```

```

        jButton4_mouseClicked(e);
    }
});
jLabel5.setFont(new java.awt.Font("Dialog", 0, 16));
jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
jLabel5.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel5.setText("to display gridded data from local and remote datasets.");
jLabel6.setFont(new java.awt.Font("Dialog", 0, 16));
jLabel6.setAlignmentX((float) 0.5);
jLabel6.setAlignmentY((float) 1.0);
jLabel6.setMaximumSize(new Dimension(300, 60));
jLabel6.setMinimumSize(new Dimension(300, 60));
jLabel6.setPreferredSize(new Dimension(470, 30));
jLabel6.setToolTipText("");
jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
jLabel6.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel6.setText("The Interactive Sounding stand-alone application allows users");
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar, BorderLayout.WEST);
contentPane.add(statusBar, BorderLayout.SOUTH);
contentPane.add(jTabbedPane1, BorderLayout.CENTER);
jPanel3.add(jButton2, new PaneConstraints("jButton2", "jButton2",
PaneConstraints.ROOT, 0.5f));
jPanel3.add(jLabel2, new PaneConstraints("jLabel2", "jButton2",
PaneConstraints.TOP, 0.4407583f));
jPanel3.add(jLabel1, new PaneConstraints("jLabel1", "jLabel2",
PaneConstraints.TOP, 0.47500002f));
jTabbedPane1.add(jPanel1, "Gridded Data Viewer");
jPanel2.add(jLabel6, new PaneConstraints("jLabel6", "jLabel6",
PaneConstraints.ROOT, 0.5f));
jPanel2.add(jButton4, new PaneConstraints("jButton4", "jLabel6",
PaneConstraints.BOTTOM, 0.77669907f));
jPanel2.add(jLabel5, new PaneConstraints("jLabel5", "jButton4",
PaneConstraints.TOP, 0.2863248f));
jTabbedPane1.add(jPanel3, "Image Viewer");
jTabbedPane1.add(jPanel2, "Interactive Sounding");
jPanel1.add(jLabel3, new PaneConstraints("jLabel3", "jLabel3",
PaneConstraints.ROOT, 0.5f));
jPanel1.add(jButton3, new PaneConstraints("jButton3", "jLabel3",
PaneConstraints.BOTTOM, 0.8341232f));

```

```

    jPanel1.add(jLabel4, new PaneConstraints("jLabel4", "jButton3",
PaneConstraints.TOP, 0.3579545f));
}
/**File | Exit action performed*/
public void jMenuItemFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
}
/**Help | About action performed*/
public void jMenuItemHelpAbout_actionPerformed(ActionEvent e) {
    Frame1_AboutBox dlg = new Frame1_AboutBox(this);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
}
/**Overridden so we can exit when window is closed*/
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        jMenuItemFileExit_actionPerformed(null);
    }
}

/**Action to perform when GDV button is clicked*/
void jButton3_mouseClicked(MouseEvent e)
{
    try
    {
        jButton3.setBackground(Color.green);
        makeGDV();
    }
    catch (Exception ex)
    {
        System.out.println("Caught " + ex.toString());
    }
}

/**Construct a Gridded Data Viewer (GDV) Application*/
void makeGDV()
{
    try
    {

```

```

        makeGridViewer top_jf = new makeGridViewer();
    }
    catch (Exception ex)
    {
        System.out.println("Caught " + ex.toString());
    }
}

/**Action to perform when Image Viewer button is clicked*/
void jButton2_mouseClicked(MouseEvent e)
{
    try
    {
        jButton2.setBackground(Color.green);
        makeImageViewer();
    }
    catch (Exception ex)
    {
        System.out.println("Caught " + ex.toString());
    }
}

/**Construct an Image Viewer Application*/
void makeImageViewer()
{
    try
    {
        boolean do3d = !((System.getProperty("ImageViewer.mode",
"2D")).equalsIgnoreCase("2D"));

        ImageViewer iv = new ImageViewer("Satellite Image Viewer", do3d);
        iv.setVisible(true);
    }
    catch (Exception ex)
    {
        System.out.println("Caught " + ex.toString());
    }
}

/**Action to perform when Sounding button is clicked*/
void jButton4_mouseClicked(MouseEvent e)
{
    jButton4.setBackground(Color.green);
    makeSounding();
}

```

```
/**Construct a Sounding Application*/  
void makeSounding()  
{  
    try  
    {  
        new InterSounding().show();  
    }  
    catch (VisADException visex)  
    {  
        System.out.println("Caught " + visex);  
    }  
    catch (RemoteException remex)  
    {  
        System.out.println("Caught " + remex);  
    }  
}  
}
```

C.2 Frame1_AboutBox.java Source Code

```
package application1;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * Title:          iMetApps Experimental Application - Design 1
 * Description:    This is an experimental application using design 1. This experimental
 *                application uses Unidata MetApps prototype stand-alone applications
 *                GDV.jar, SoundingApplication.jar, and ImageViewer.jar.
 * Created:       December 2001
 * Company:      AFIT/ENP
 * @author       Captain Harmen P. Visser
 * @version      1.0
 */

public class Frame1_AboutBox extends JDialog implements ActionListener {

    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();
    JPanel insetsPanel1 = new JPanel();
    JPanel insetsPanel2 = new JPanel();
    JPanel insetsPanel3 = new JPanel();
    JButton button1 = new JButton();
    JLabel imageLabel = new JLabel();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    FlowLayout flowLayout1 = new FlowLayout();
    GridLayout gridLayout1 = new GridLayout();
    String product = "iMetApps Experimental Application - Design 1";
    String version = "1.0";
    String copyright = "Copyright (c) 2001";
    String comments = "This is an experimental application using design 1. This
experimental application uses Unidata MetApps prototype stand-alone applications
GDV.jar, SoundingApplication.jar, and ImageViewer.jar.";
    public Frame1_AboutBox(Frame parent) {
```

```

super(parent);
enableEvents(AWTEvent.WINDOW_EVENT_MASK);
try {
    jbInit();
}
catch(Exception e) {
    e.printStackTrace();
}
pack();
}
/**Component initialization*/
private void jbInit() throws Exception {
    //imageLabel.setIcon(new ImageIcon(Frame1_AboutBox.class.getResource("[Your
Image]")));
    this.setTitle("About");
    setResizable(false);
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
    button1.setText("Ok");
    button1.addActionListener(this);
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    this.getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
}
/**Overridden so we can exit when window is closed*/
protected void processWindowEvent(WindowEvent e) {

```

```

    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        cancel();
    }
    super.processWindowEvent(e);
}
/**Close the dialog*/
void cancel() {
    dispose();
}
/**Close the dialog on a button event*/
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button1) {
        cancel();
    }
}
}
}

```


C.4 interSounding.java Source Code

```
package application1;

import java.awt.*;
import java.awt.event.*;
import java.rmi.RemoteException;
import javax.swing.*;
import visad.VisADException;
import ucar.unidata.view.sounding.*;

/**
 * Title:      interSounding.java
 * Description: This class is used to build an Interactive Sounding window from within
               iMetApps
 *
               Experimental Application - Design 1. This class is necessary to handle
               exceptions
               thrown by VisAD and the precompiled Sounding Application files.
 * Created:    December 2001
 * Company:    AFIT/ENP
 * @author     Captain Harmen P. Visser
 * @version    1.0
 */

public class InterSounding extends JFrame
{
    public InterSounding() throws VisADException, RemoteException
    {
        super("Interactive Sounding Application");

        /**This listener allows sounding to be closed and iMetApps Control Panel to
        remain open*/
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e){}
        });
        new SoundingApp(getRootPane(), true);
        pack();
        Dimension screenSize = getToolkit().getScreenSize();
        Dimension frameSize = getSize();
        setLocation((screenSize.width - frameSize.width)/2,(screenSize.height -
        frameSize.height)/2);
    }
}
```

C.5 makeGridViewer.java Source Code

```
package application1;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.JComponent.*;
import ucar.unidata.ui.*;
import ucar.unidata.util.*;
import ucar.unidata.gridviewer.*;

/**
 * Title:      makeGridViewer.java
 * Description: The classes contained in this file are an adaptation of the Main.java,
 *              v 1.22 2001/05/01 15:21:07, Copyright 1997-2000 Unidata Program
 *              Center/University Corporation for Atmospheric Research, P.O. Box 3000,
 *              Boulder, CO 80307, support@unidata.ucar.edu. It is part of the MetApps
 *              library. This library is free software and can be redistributed and
 *              modified under the terms of the GNU Lesser General Public License as
 *              published by the Free Software Foundation; either version 2.1 of the
 *              License, or (at your option) any later version.
 *              This file (containing two classes) provides a way to create a Gridded
 *              Data Viewer (GDV) Application via the iMetApps launchpad.
 * Created:    December 2001
 * Company:    AFIT/ENP
 * @author     Captain Harmen P. Visser
 * @version    1.0
 */

public class makeGridViewer implements TopLevel
{
    JFrame top;
    Common common;

    makeGridViewer()
    {
        top = new JFrame("Gridded Data Viewer");

        top.addWindowListener(new WindowAdapter() {});

        common = new Common(top, this, false);
    }
}
```

```

        top.pack();
        top.setVisible(true);
    }

    public RootPaneContainer getRootPaneContainer()
    {
        return top;
    }

    public boolean isApplet()
    {
        return false;
    }

    public void close()
    {
        save();
        System.exit(0);
    }

    public void save()
    {
        common.saveConfig();
    }
}

class Common
{
    SerializedObjectStore store = new SerializedObjectStore("metapps", "GridViewer",
"main");
    private Controller control;
    private UI ui;
    private Component mainC;

    Common(Component mainC, TopLevel topLevel, boolean isApplet)
    {
        this.mainC= mainC;
        ucar.unidata.util.Debug.fetchPersistentData( store);

        if (Debug.isSet("util.showProperties"))
        {
            try
            {
                Properties p = System.getProperties();

```

```

Enumeration enum = p.keys();

while (enum.hasMoreElements())
{
    Object key = enum.nextElement();
    System.out.println(" "+key + " = " + p.get(key));
}
}
catch (SecurityException e)
{
    System.out.println("not allowed to get Properties");
}

ucar.unidata.ui.Help.setTopDir("/auxdata/java/help/GDV");
}

control = new Controller(topLevel, store);
ui = new UI( topLevel, store, control);
control.setUI( ui);

control.addMapBean( new ucar.unidata.gis.worldmap.WorldMapBean());
control.addMapBean( new
ucar.unidata.gis.shapefile.ShapeFileBean("/auxdata/maps/Countries.zip"));
control.addMapBean( new ucar.unidata.gis.shapefile.ShapeFileBean());
}

void saveConfig()
{
    control.storePersistentData();
    ui.storePersistentData();
    store.put("MainWindowBounds", mainC.getBounds());
    ucar.unidata.util.Debug.storePersistentData( store);
    store.save();
}
}

```

Bibliography

- Army Test and Evaluation Command (ATEC), 2001: *4DWX Weather Forecasting Technology*. <https://www.4dwx.org/>
- Apache Software Foundation, 2001: *Apache XML Project*. <http://xml.apache.org/>
- Bowers, J., 2000: Development of Operational Meso-gamma-Scale Numerical Weather-Prediction System for Army Test Ranges. *Battlespace Atmospheric and Cloud Impacts on Military Operations (BACIMO) Conference 2000*. Fort Collins CO.
- Caron, J., 1999: Component Based Software for Scientific Application Development. *Proceedings of the Fifteenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, American Meteorological Society.
- Center for Ocean-Land-Atmosphere Studies (COLA), 2001: *GrADS Home Page*. <http://grads.iges.org/grads/>
- Data Assimilation Office (DAO), National Aeronautics and Space Administration (NASA), 2001: *GrADS with Athena Widgets*. http://dao.gsfc.nasa.gov/software/grads/win32/latest/pc-dist/doc/gagui/gagui_intro.html
- Fowler, M. and K. Scott, 2000: *UML Distilled (2nd Edition): A Brief Guide to the Standard Object Modeling Language*. Reading MA: Addison Wesley, 185 pp.
- Gamma, E., and others, 1995: *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison Wesley, 395 pp.
- Hunter, J. and B. McLaughlin, 2001: *JDOM Web Site*. <http://jdom.org>
- Lindholm, D., 2001: *Why was VisAD chosen as the basis for the 4DWX next generation display (VMET): An engineering perspective*. http://atec-server.rap.ucar.edu/vmet/why_visad.html
- Microsoft, 2001: *Microsoft DirectX Home Page*. <http://www.microsoft.com/directx/>
- National Center for Atmospheric Research (NCAR), 2000. *Research Applications Program: Information Technology -- Database Management*. <http://www.rap.ucar.edu/technology/it/>

- National Center for Atmospheric Research (NCAR), 2001. *VMET: Visual Meteorology Tool*. <https://atec-server.rap.ucar.edu/vmet/>
- Pressman, R.S., 2000: *Software Engineering: A Practitioner's Approach, Fifth Edition*. Boston: McGraw-Hill Higher Education, 888 pp.
- Rew, R.K., and others, 1997: *NetCDF User's Guide for C*. <http://www.unidata.ucar.edu/packages/netcdf/guidec/>
- Schatzman, J.C., 2001: Writing High Performance Java Code Which Runs as Fast as Fortran, C or C++. *Proceedings of SPIE, Vol. 4521: Java/Jini Technologies*, SPIE Press, pp. 106-114.
- Schildt, H., 2000: *Java™ 2: The Complete Reference, Fourth Edition*. Berkeley CA: McGraw-Hill Professional Publishing, 1040 pp.
- Shalloway, A., and J.R. Trott, 2002: *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Reading MA: Addison Wesley, 368 pp.
- Silicon Graphics Incorporated (SGI), 2001: *OpenGL: The 3D Standard*. <http://www.opengl.org/users/about/index.html#where>
- SourceForge, 2001: *Vis5d+ Home Page*. <http://vis5d.sourceforge.net/>
- Space Science and Engineering Center (SSEC), 1997: *A Brief History of McIDAS*. http://www.ssec.wisc.edu/software/mcidas_history.html
- Space Science and Engineering Center (SSEC), 2000: Vis5D Version 5.2 README File <ftp://www.ssec.wisc.edu/pub/vis5d-5.2/README>
- Space Science and Engineering Center (SSEC), 2001: *VisAD Home Page*. <http://www.ssec.wisc.edu/~billh/visad.html>
- Telfeyan, B., 2001: Chief, Technology Exploitation Branch (DNXT), Air Force Weather Agency (AFWA), Offutt AFB NE. Personal communication over the course of thesis proposal preparation.
- Unidata MetApps Discussion Area (UMADA), 2001: *UMADA Home Page*. <http://www.unidata.ucar.edu/projects/umada/index.html>
- University Corporation for Atmospheric Research (UCAR), 1997: *Unidata: 2003. A Proposal to the National Science Foundation*. <http://www.unidata.ucar.edu/proposals/NSF2003/>

University Corporation for Atmospheric Research (UCAR), 2000: *Unidata MetApps Project*. <http://www.unidata.ucar.edu/projects/metapps/index.html>

University Corporation for Atmospheric Research (UCAR), 2001: *Unidata McIDAS Home Page*. <http://www.unidata.ucar.edu/packages/mcidas/>

Vlissides, J.M., and others, 1996: *Pattern Languages of Program Design 2*. Reading, MA: Addison Wesley, 605 pp.

Woo, M. and others, 1999: *OpenGL Programming Guide (3rd Edition): The Official Guide to Learning OpenGL, Version 1.2*. Reading MA: Addison Wesley, 800 pp.

Vita

Captain Harmen P. Visser graduated from Piper High School in Sunrise, Florida, in 1983. He entered undergraduate studies at Broward Community College, Florida where he graduated with an Associate of Arts degree in Engineering in May 1986. In 1989 he enlisted in the US Air Force and served at Barksdale AFB, Louisiana, as a Dental Laboratory Technician. In 1993 he was accepted into the Airman Education and Commissioning Program and was sent to Florida State University where he graduated with Bachelor of Science degree in Meteorology in 1996.

His first meteorological assignment was to the 46th Weather Squadron at Eglin AFB, Florida as a Wing Weather Officer. At Eglin, he served as the Northwest Florida region radar coordinator. In 1998, he was assigned to the 65th Operations Support Squadron, Lajes Field, Portugal, where he served as the Meteorological Satellite Coordinator. While stationed at Lajes, he filled the role of Logistics Group representative to the base Year 2000 Operations Center. In August 2000, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the 88th Weather Squadron, Wright-Patterson AFB, Ohio.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 074-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>						
1. REPORT DATE (DD-MM-YYYY) 08-03-2002		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From – To) Jun 2001 – Mar 2002		
4. TITLE AND SUBTITLE SUITABILITY OF UNIDATA METAPPS FOR INCORPORATION IN PLATFORM-INDEPENDENT USER-CUSTOMIZED AVIATION WEATHER PRODUCTS GENERATION SOFTWARE				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Visser, Harmen P., Captain, USAF				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GM/ENP/02M-09		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFWA/DNXT Attn: Mr. Bruce Telfeyan 106 Peacekeeper Drive Offutt AFB, NE 68113-4039 DSN: 271-1690 e-mail: bruce.telfeyan@afwa.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S)		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT <p>Due to multiple factors, including an increase in military operations tempo and the improved resolution of meteorological models, demand for access to customized aviation weather products has increased exponentially. This has given rise to a need for a multi-purpose interactive aviation weather product generation software solution. This software solution must be platform-independent, multiple data source access configurable, robust, extensible or upgradeable, user-friendly, and an improvement over current visualization applications used in the operational military aviation weather community. The purpose of this thesis is to determine whether Unidata MetApps meets these criteria.</p> <p>A software reuse and component-based engineering approach was taken in this thesis. Two experimental applications were constructed using a software design approach resembling the Facade software design pattern. The first application used existing MetApps stand-alone prototype applications, while the second exploited capabilities of the MetApps component library. Both experimental applications were measured against the above set of criteria to determine their suitability for incorporation in platform-independent user-customized aviation weather products generation software. The results prove that a Facade software design approach can be effectively used to build applications. It was determined however that, even though MetApps shows promise, it may not be suitable for incorporation into an operational application.</p>						
15. SUBJECT TERMS Meteorology, Interactive Graphics, Applications, Java, Unidata, MetApps						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy M. Jacobs, Lt Col, USAF (ENG)	
U	U	U	UU	103	19b. TELEPHONE NUMBER (Include area code) (937) 255-6565, ext 4279; e-mail: timothy.jacobs@afit.edu	

